|  |  |
| --- | --- |
|  |  |

# GPU Unterstützung für TensorFlow auf Windows 10

## Ein paar ermutigende Worte vorab

Ich hatte noch nie ein Notebook, das so glatt und konsistent funktioniert hat, wie mein neues 15 Zoll Microsoft Surface Book 2. Wirklich cooles Zeug. Es ist schade, dass all diese Sätze mit einem "Aber" enden. Hier ist das "Aber" dieser Geschichte:

Wenn man beabsichtigt, Deep Neuronal Networks DNN zu implementieren und zu optimieren, müssen die Berechnungen unbedingt auf der GPU stattfinden. Natürlich können die Berechnungen auch auf der CPU laufen, was sich einfach installieren lässt. Der Nachteil hiervon ist jedoch, dass es sehr lange dauert, bis größere Modelle trainiert sind. Das Geheimnis hinter ausgereifter KI ist, dass die perfekte Konfiguration eines Modells erst nach mehreren Trainingseinheiten erreicht ist – was viel Rechenleistung und damit Zeit beansprucht.

Wenn man KI-Gurus wie François Chollet (den Hersteller des Keras-Frameworks) fragt, würden sie sagen: "Verwende Linux- oder GPU-Cloud-Computer!" Ich muss diesen ihnen Recht geben; die Installation der GPU-Unterstützung ist auf Linux wirklich einfach und auf Windows super kompliziert.

Nichtsdestotrotz ist es möglich, Deinen Windows 10 zu zwingen, die Tore zur GPU-API zu öffnen, und ich werde in diesem Blogbeitrag zeigen, wie genau das geht. Danach kannst Du Deine neuronalen Netze viel schneller trainieren und holst die Zeit locker wieder rein, die Du jetzt investieren musst.

Zu Beginn ein wichtiger Hinweis: Am Anfang dachte ich, dass ich die empfohlenen Treiber- und Softwareversionen ignorieren kann und habe immer die aktuellsten Versionen genommen. Das war eine wirklich schlechte, frustrierende und zeitraubende Idee. Sobald ich die Versionen verwendet hatte, die ich auch in diesem Beitrag empfehle, war es viel einfacher.

Also hol dir einen Kaffee, leg dir coole Musik auf die Ohren und starte durch 😊

## Get Python into shape

Zuerst ist es sinnvoll, die Python-Umgebung in eine vordefinierte Form zu bringen.

Wenn man Python nicht bereits installiert hat, kann dies über die Anaconda-Distribution (https://www.anaconda.com/download/#windows) getan werden. Bei dieser Installation sollte man darauf achten, das Bundle für Python 3.xx 64 Bit, und nicht 2.xx zu verwenden.

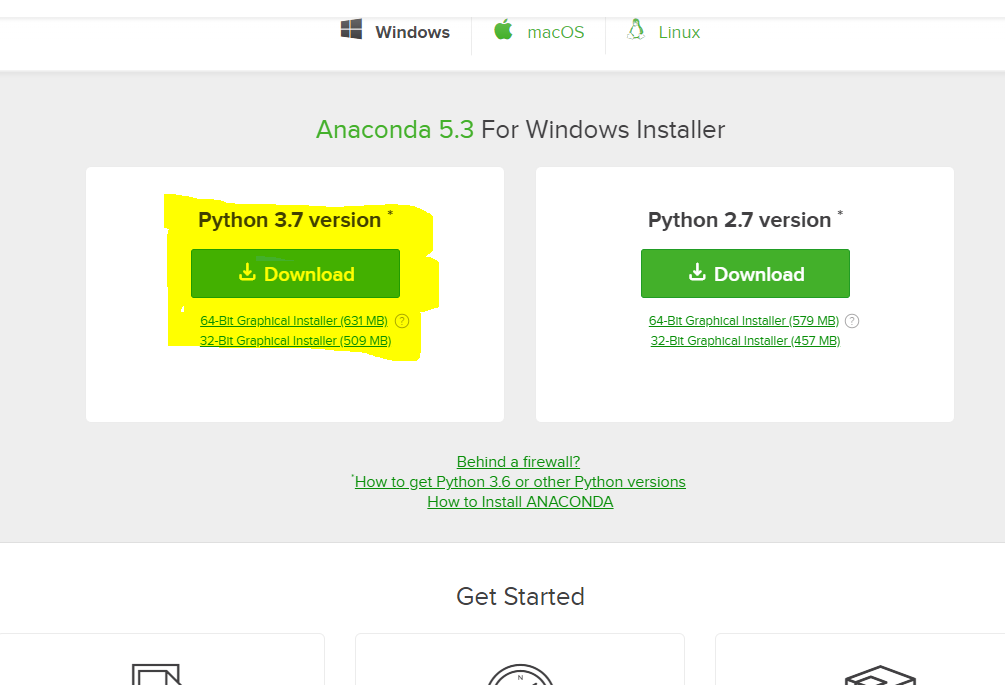


Abb 1: Download Anaconda für Python 3.7

Zunächst muss pip (pip ist der Python-Paketierungsmanager) auf die aktuellste Version aktualisiert werden:

1. Starte die **cmd** als Administrator
2. Run “**python.exe -m pip install --upgrade pip**”
3. Run “**pip –version**” zum prüfen der Version. Bei mir ist es “**pip 18.1 from c:\programdata\anaconda3\lib\site-packages\pip (python 3.6)**”

Ich habe alle TensorFlow und Keras Bibliotheken deinstalliert, wenn sie nicht Version 1.10.0 sind. Kein Witz: Ich empfehle, genau die genannte Version zu verwenden, um Zeit zu sparen. Ansonsten passen die Kombinationen nicht.

Run “**pip show tensorflow tensorflow-gpu keras**”

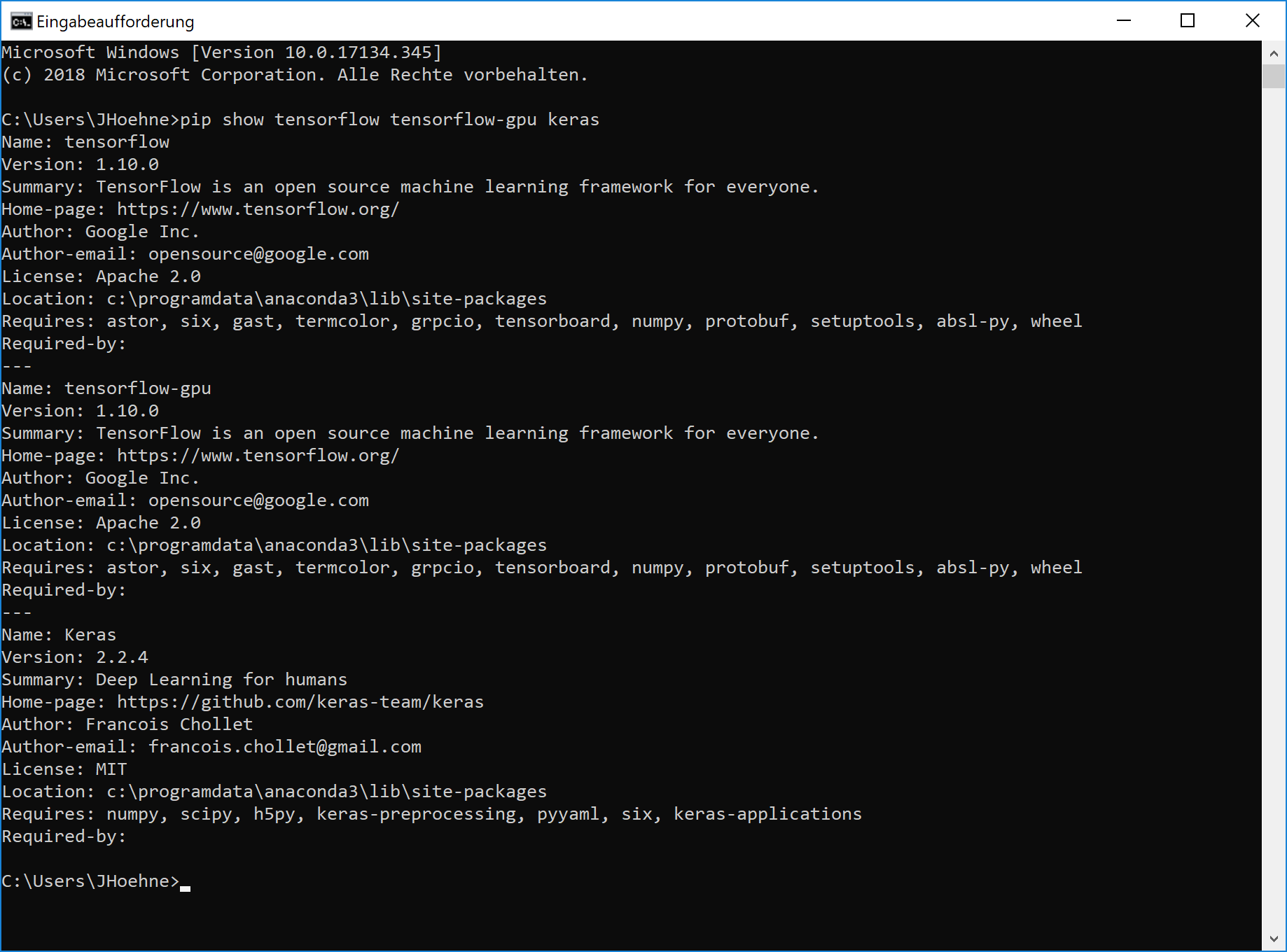


Abb 2: Überprüfen, ob und welche Version von Tensorflow insatlliert ist.

Keine Sorge, wenn TensorFlow noch nicht installiert ist, das werden wir später noch tun.

## Installiere Microsoft Visual Studio 2017

Um genau zu sein: Visual Studio brauche ich eigentlich nicht für die GPU-Unterstützung und kann es später wieder deinstallieren. Es ist jedoch notwendig, Visual Studio zu verwenden, um die GPU-Treiberbeispiele zu kompilieren, und zu überprüfen, ob die Installation erfolgreich war. Es ist der erste große Stopp auf unserem Weg, um testen zu können, ob der Betriebssystemteil einwandfrei funktioniert. Ich lade deshalb die "Visual Studio IDE Community Edition 2017" herunter:

<https://visualstudio.microsoft.com/>

Nun wähle ich die zu installierenden Pakete aus. Am Ende gibt es nur eine Sache, die wirklich wichtig ist. Der "**Windows 10 SDK (10.0.15063.0)**" muss installiert werden.

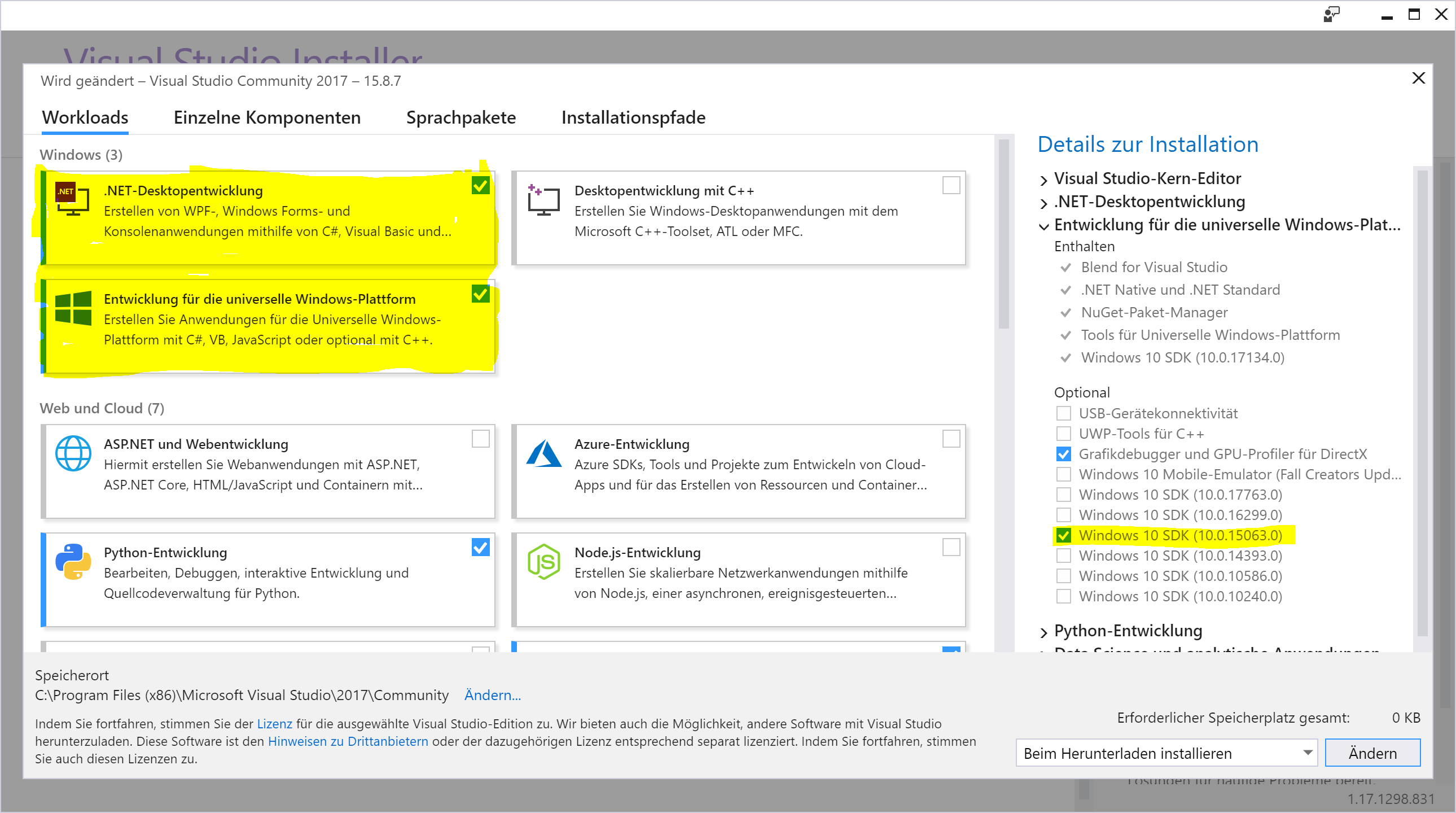


Abb 3: Alle nötigen Pakete für die Visual Studio Installation selektieren.

## Windows Update

Dein Windows sollte auf dem neuesten Stand sein. Für mich funktionierte alles viel besser, nachdem ich das letzte NVIDIA-Treiber-Update installiert hatte.

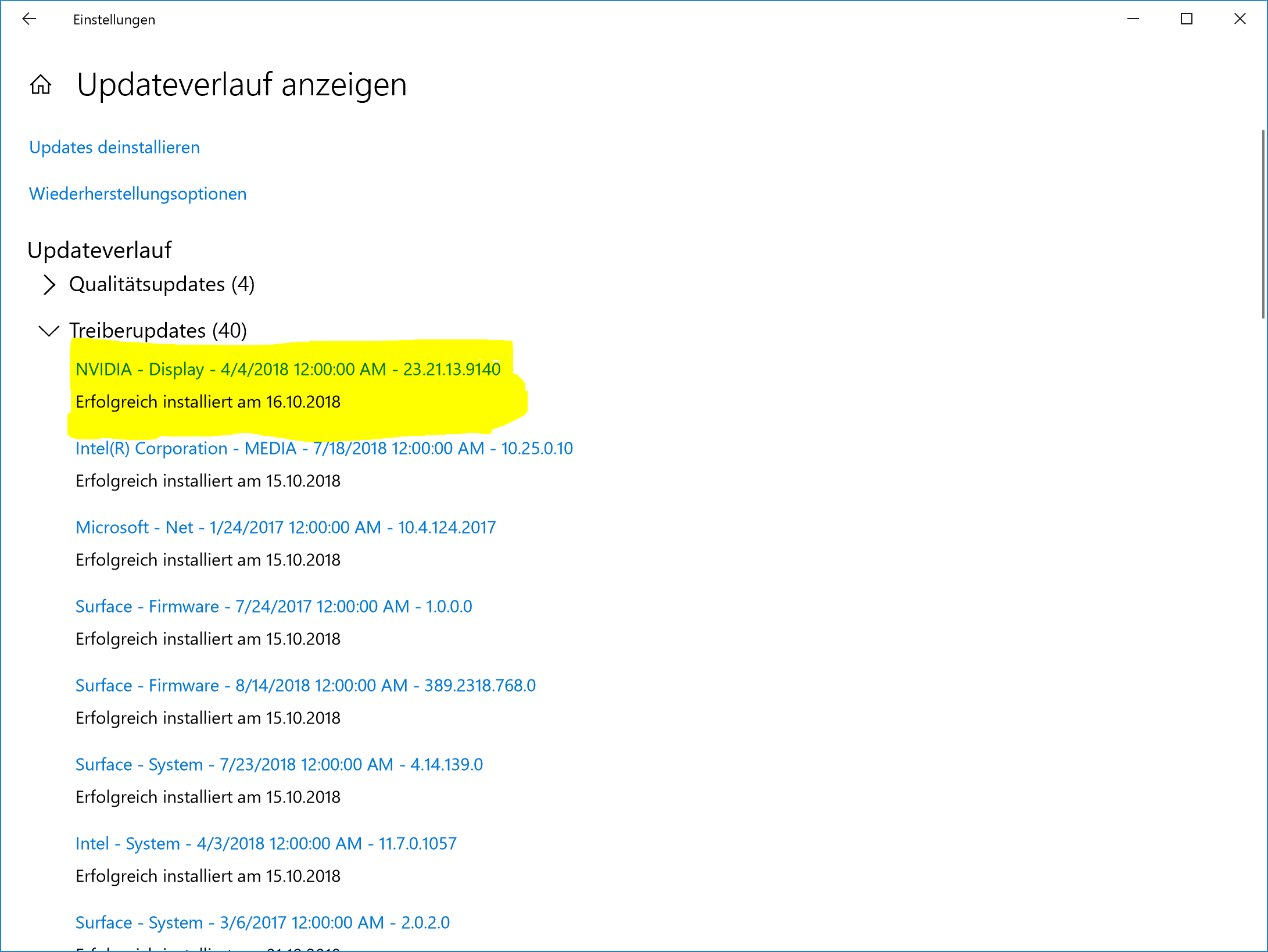
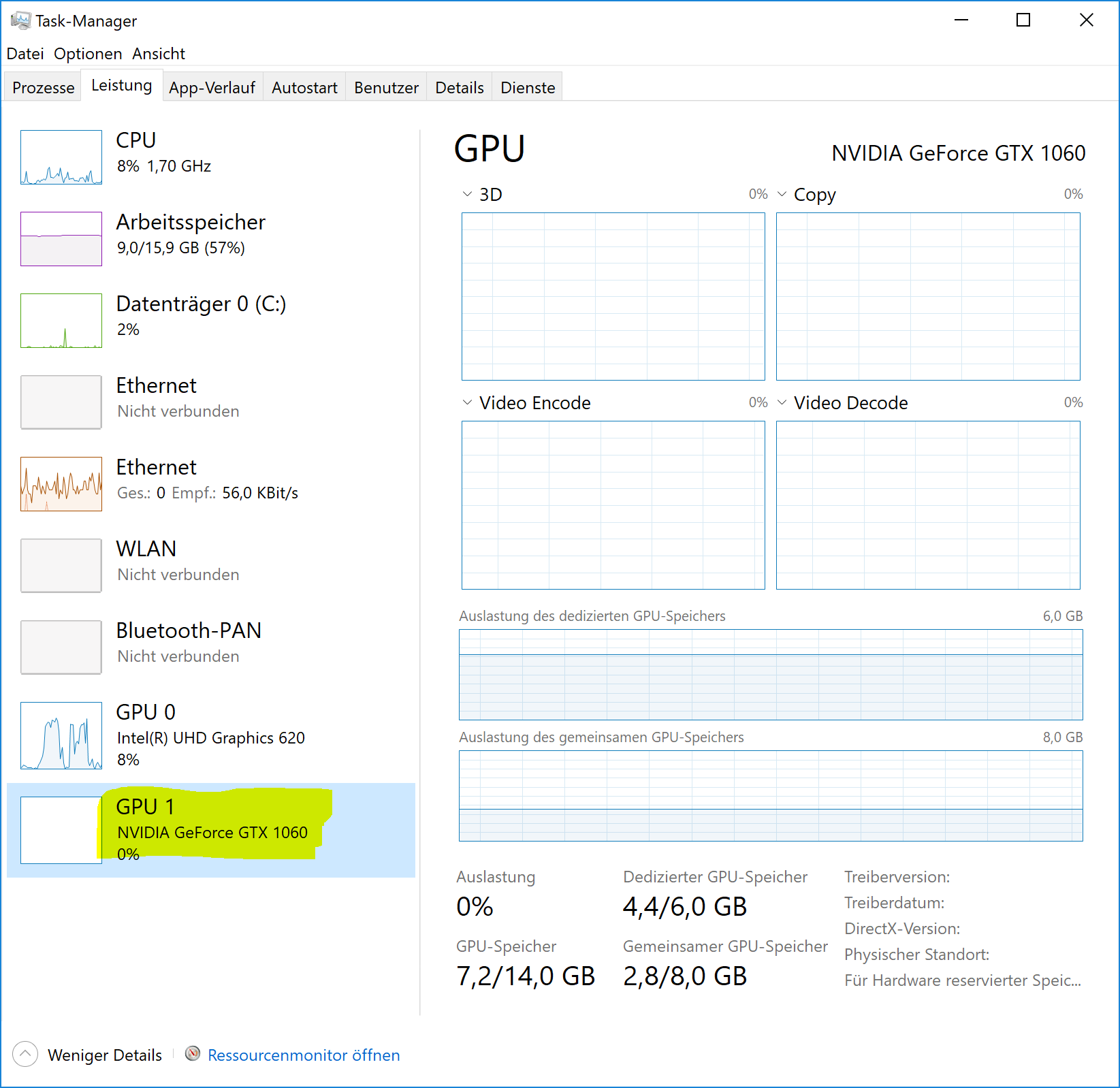


Abb. 4: Das Update des NVIDIA-Displays gibt einen Hinweis, ob der NVIDIA Grafiktreiber erfolgreich installiert wurde oder nicht.

Um herauszufinden, ob der NVIDIA Treiber richtig installiert und konfiguriert ist, kannst Du folgende Schritte machen:

Schritt 1: Öffne im Taskmanager den Reiter „Leistung“:

Abb 5: Die CPU 1 muss unter NVIDIA GeForce GTX 1060 laufen

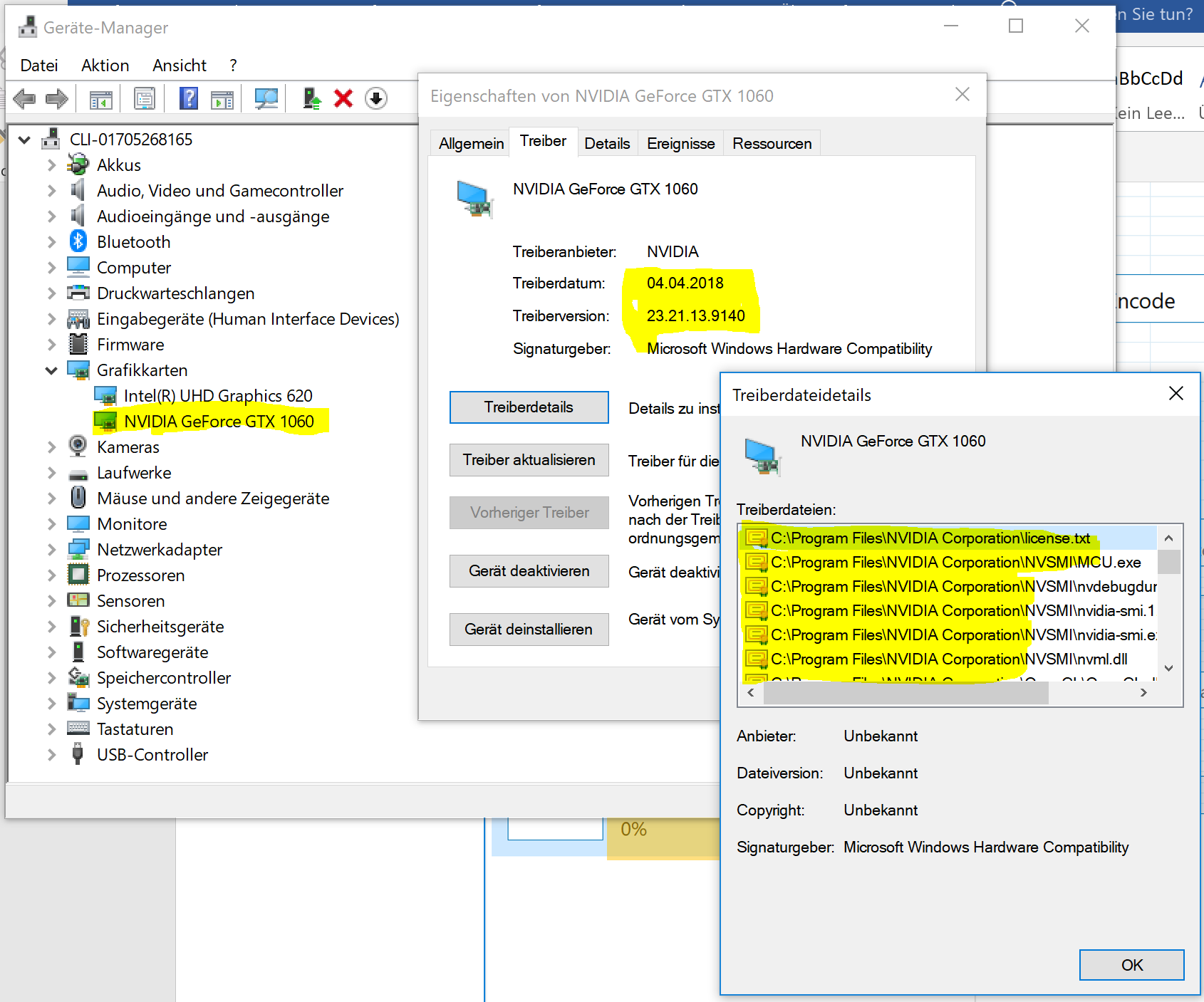


Abb. 6: Im zweiten Schritt werden die überprüft.

Der Treiber muss unter "C:\Program Files\NVIDIA Corporation\ installiert sein.

Auch hier ist es wichtig, mit den Versionsnummern pingelig zu sein. Ich habe herausgefunden, dass es nicht viel Versionstoleranz in den Tools gibt.

Im dritten Schritt prüfe ich nun die NVIDIA Einstellungen.

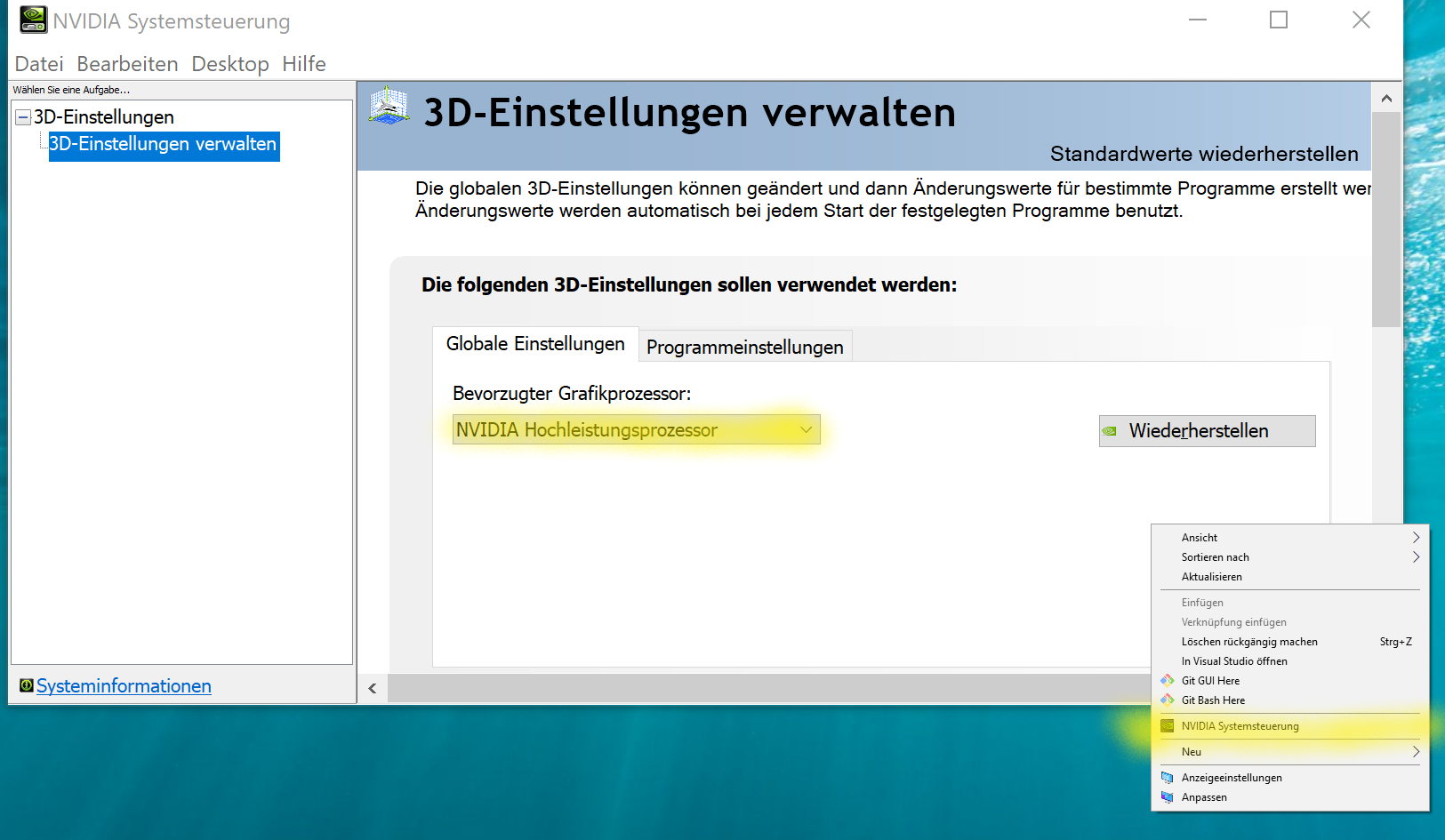


Abb 7: Der NVIDIA Hochleistungsprozessor sollte hier ausgewählt sein

Wenn man mit der rechten Maustaste auf den Desktop klickt, bekommt man den Menüpunkt "NVIDIA Systemsteuerung" (wenn nicht, dann versuche den Treiber von NVIDIA herunterzuladen und zu installieren. Das funktionierte bei mir nicht, da Windows den Treiber nach jedem Neustart entfernt hat, aber vielleicht....).

## CUDA 9.0 installieren

**CUDA** ist das API-Gateway zu Deinem NVIDIA Grafikprozessor, um C++-Zugriff darauf zu haben. Über **CUAD** kann man ein Deep Neuronal Network massiv parallel berechnen lassen. Allerdings benötigt man genau die Version "**cuda\_9.0.176.1\_windows**" und alle Patches. Es ist ein wenig schwierig, sie zu finden, da es sich bereits um ältere Versionen handelt. Einige Screenshots dazu finden sich hier:

<https://developer.nvidia.com/cuda-downloads>

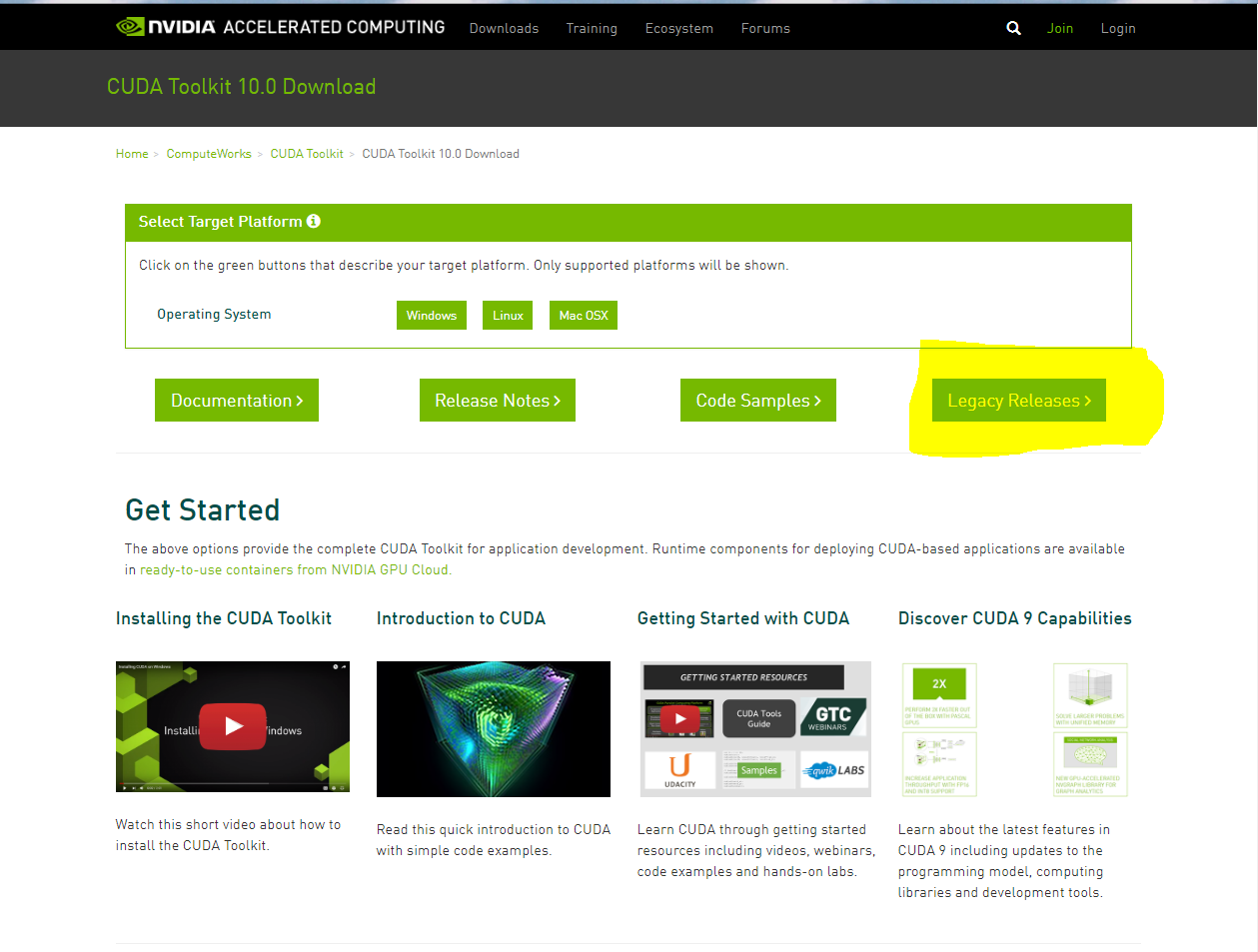


Abb 8: Hinter dem Button „Legacy Releases“ verbergen sich die älteren Versionen

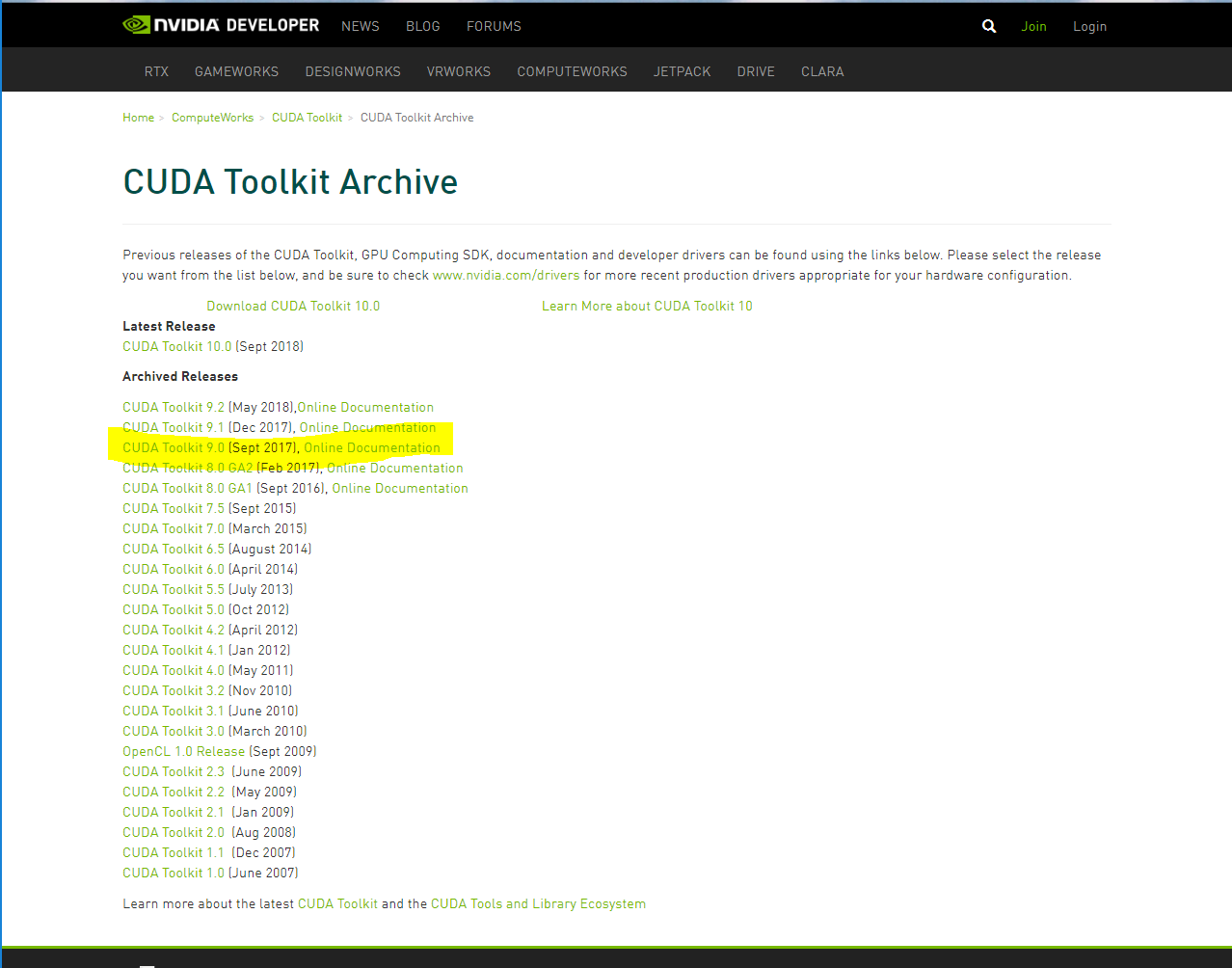


Abb 9: Hier kommt es auf die genaue CUDA Toolkit Versionsnummer an.

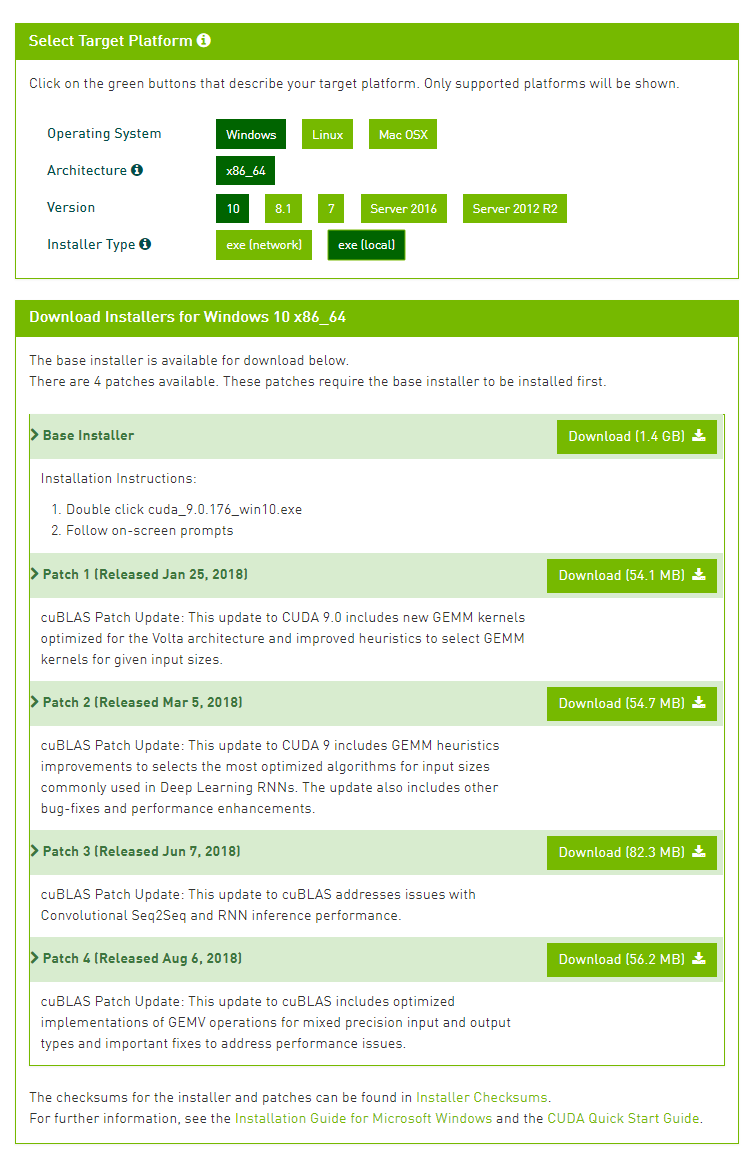


Abb 10. Den Baseinstaller und alle Patches herunterladen

Nun lädt man den **Basis-Installer** und alle **Patches** herunter.

Dann installiert man den "**Base Installer**". Das Programm warnte mich, dass es einen Versionskonflikt gibt. Ich habe diese Warnung aber ignoriert und die Installation fortgesetzt.



Abb 11: Die Sicherheitswarnung kann man ignorieren

Danach konnte ich alle Patches in aufsteigender Reihenfolge installieren.

## Installiere cuDNN

Für Deep Neuronal Networks cuDNN bauche ich die CUDA Erweiterung, und zwar genau die Version: "**cudnn-9.0-windows10-x64-v7.3.0.29**". Es ist etwas schwierig, den Download zu finden, da man zunächst ein Konto bei NVIDIA erstellen muss (Ich habe direkt ein Konto erstellt und nicht meinen Google oder Facebook Account genutzt, da der Login so definitiv funktioniert).

<https://developer.nvidia.com/cudnn>

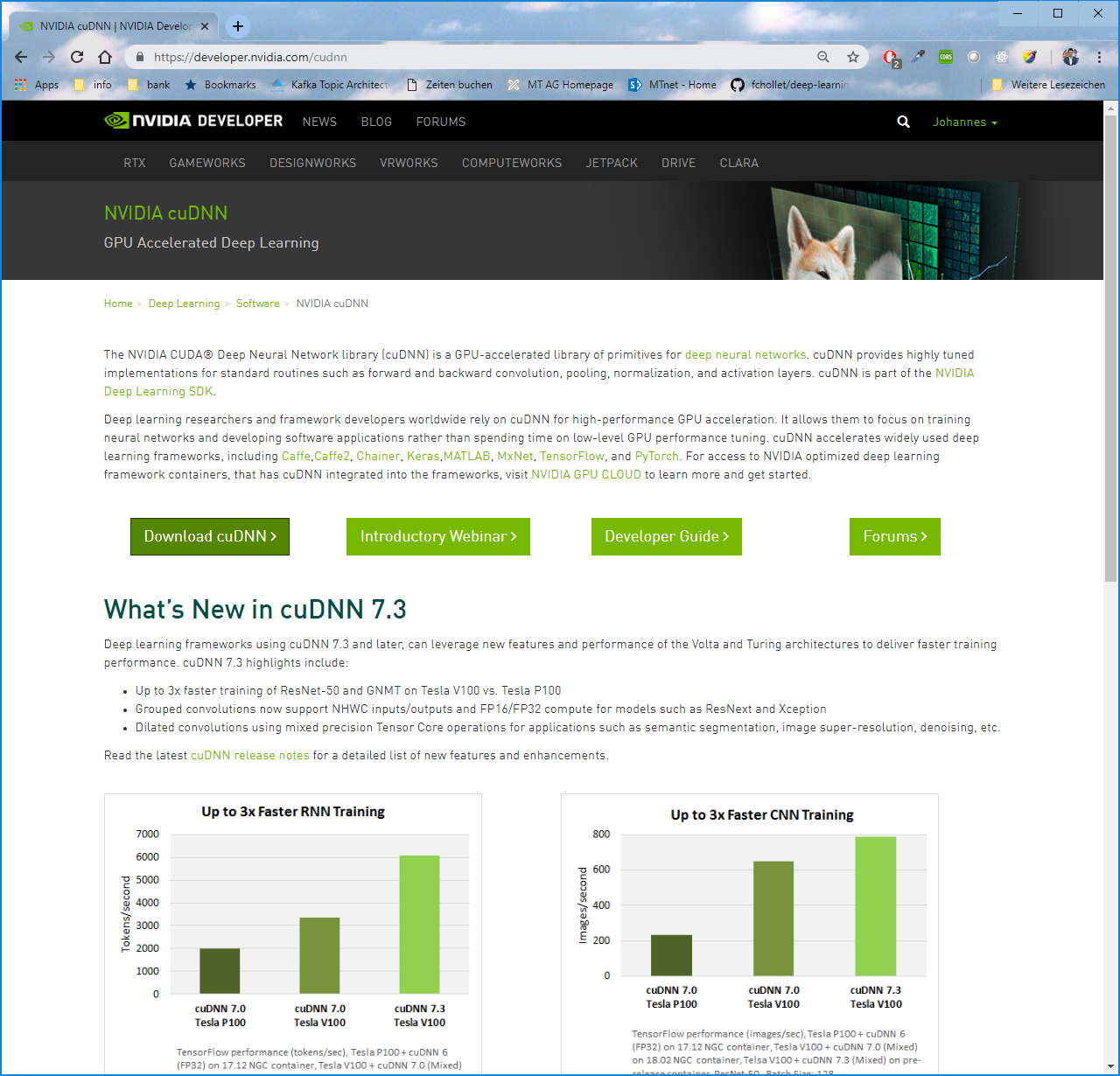


Abb 12: cuDNN Download Page, nachdem man sich angemeldet hat.

Klicke auf „**Download cuDNN**“

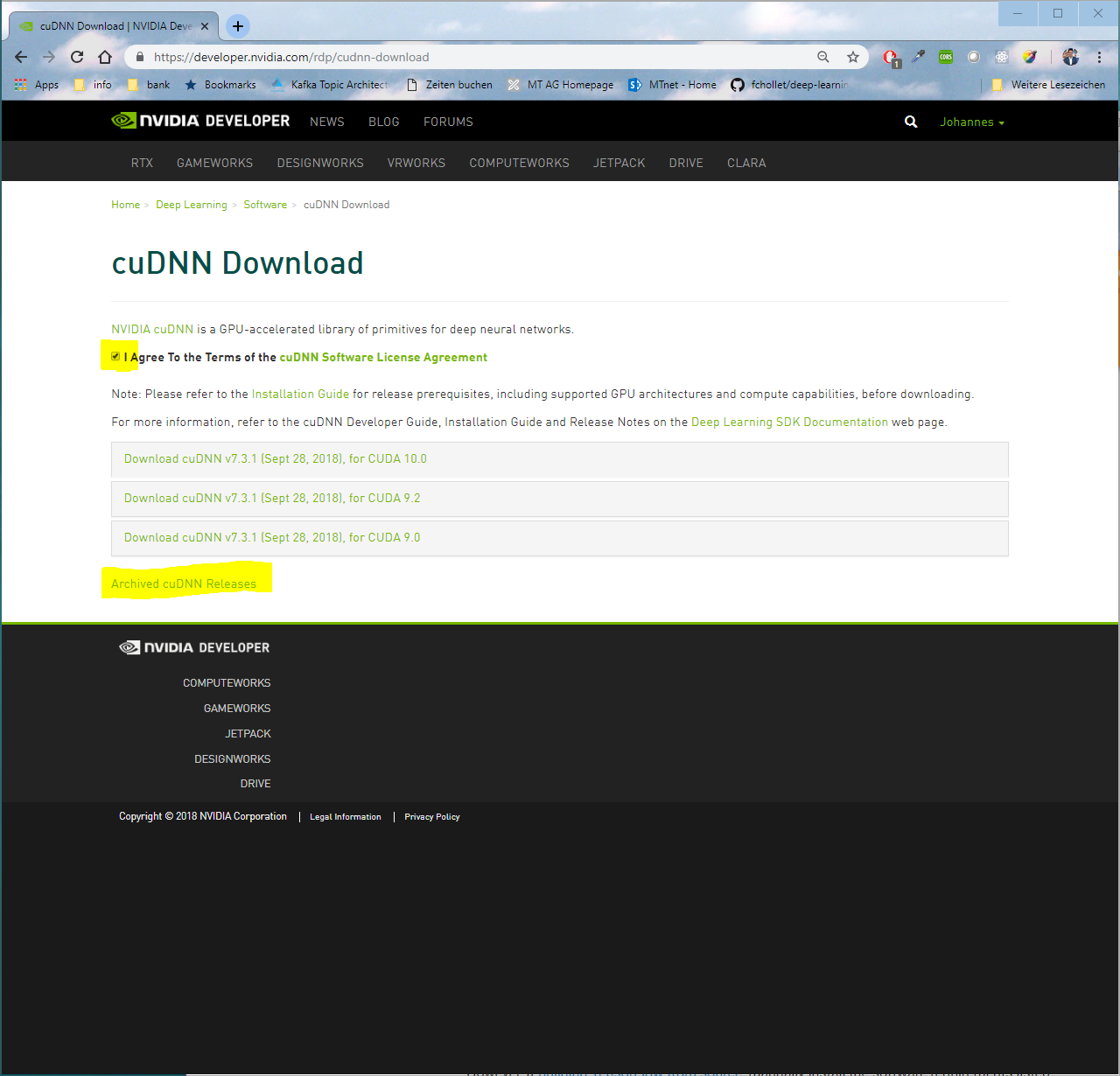


Abb 13: cuDNN Download Page nachdem man den Haken vor „I Agree…“ gesetzt hat

Ich selektiere hier zunächst „**I Agree To the Terms oft he…**“. Danach wird der Link „**Archived cuDNN Releases**“ eingeblendet.

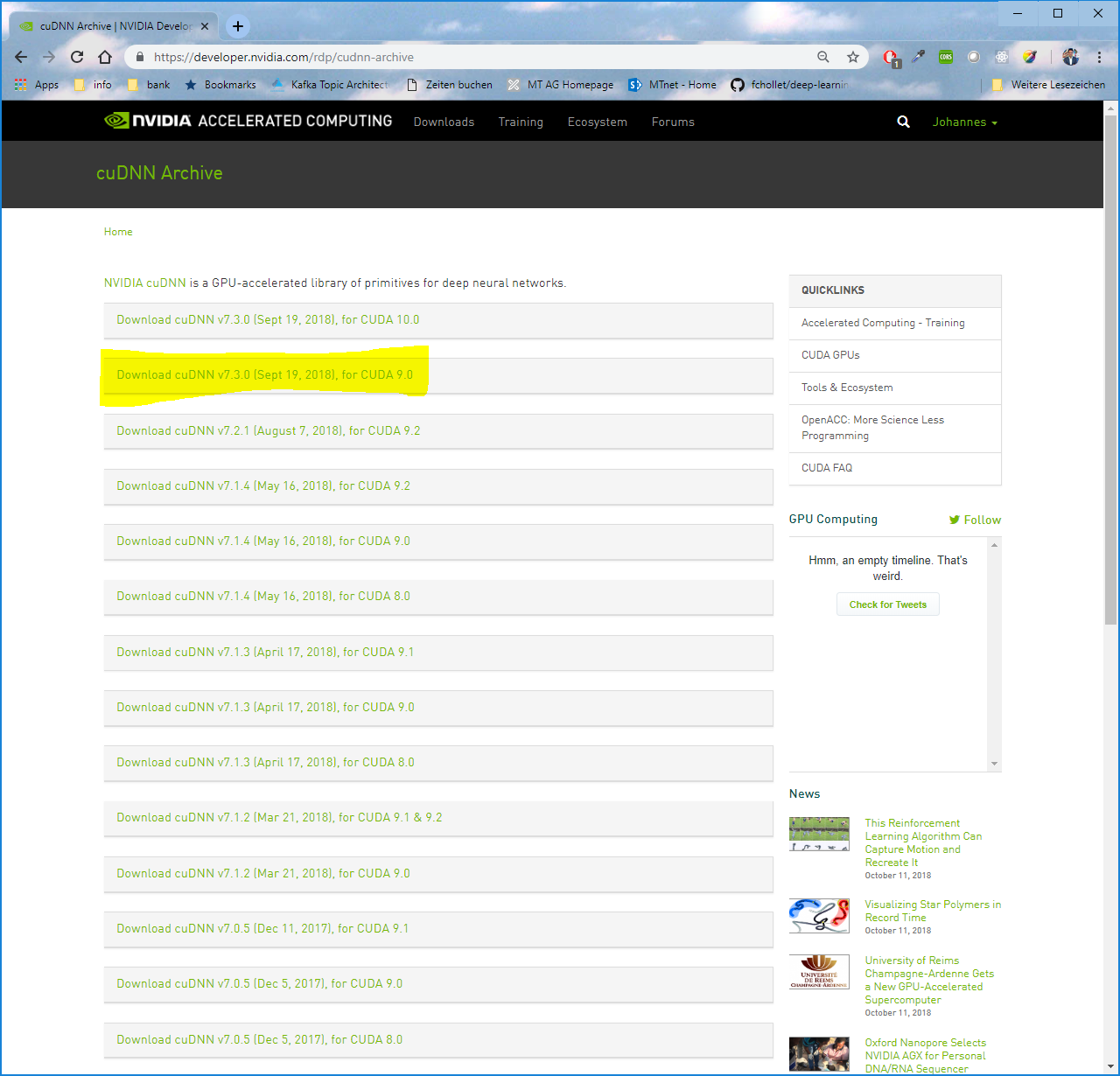
Setze 

Abb 14: Die richtige Version von cuDNN ist extrem wichtig

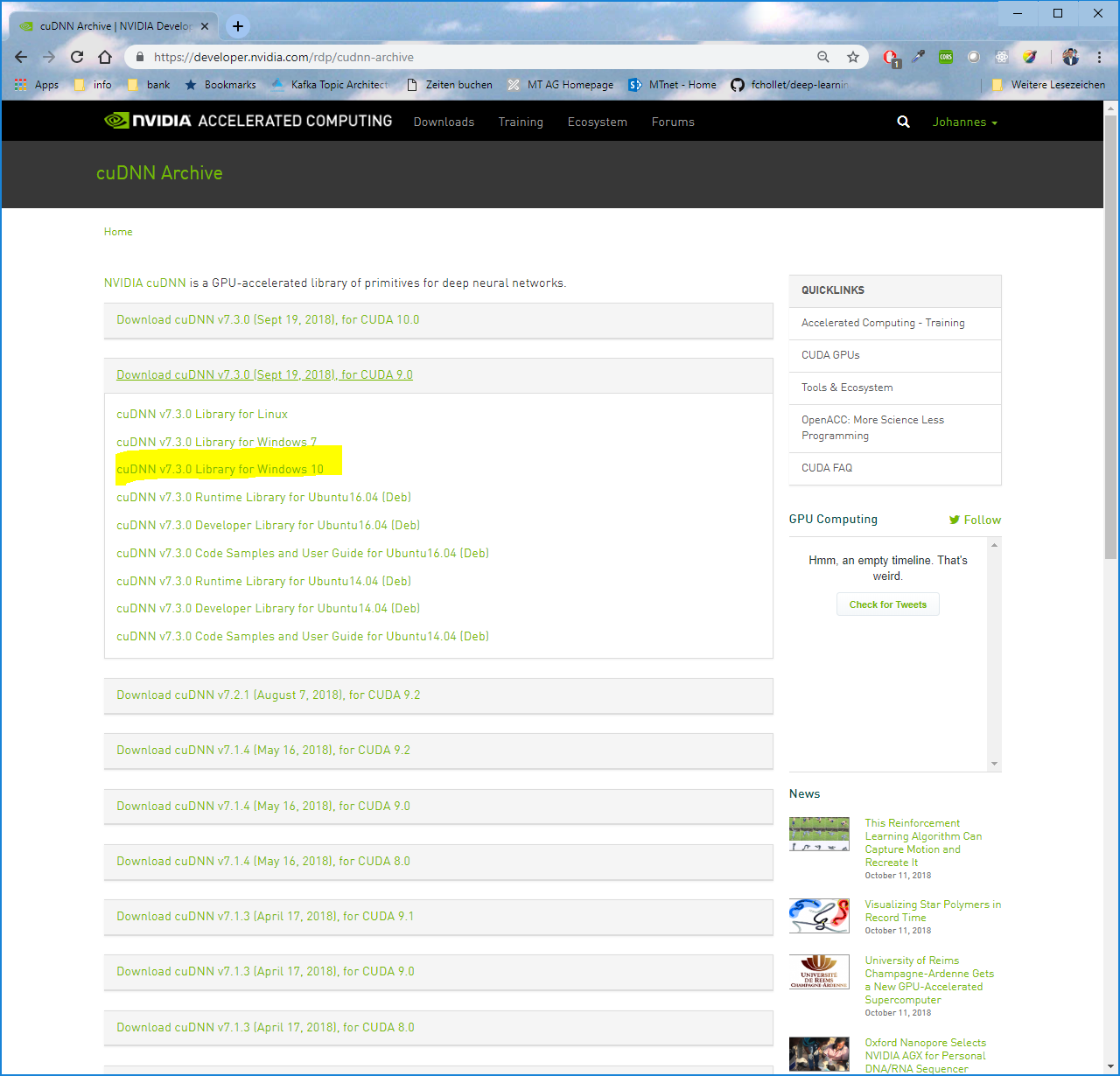


Abb 15: Download von cuDNN für Windows 10

cuDNN is eine ZIP Datei, die man einfach entpackt und den Inhalt in das CUDA Installationsverzeichnis kopiert:

1. Kopiere **.\cuda\bin\cudnn64\_7.dll** nach **C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\bin**
2. Kopiere **.\cuda\ include\cudnn.h** nach **C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\include**
3. Kopiere **.\cuda\lib\x64\cudnn.lib** nach **C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v9.0\lib\x64**

## Prüfen, ob die CUDA Installation geklappt hat

Ich prüfe nun, ob der Windows-Teil erfolgreich war, damit ich mich danach auf den reinen Python-Teil konzentrieren können. Für die Fehlersuche ist dieser kleine Zwischenschritt extrem hilfreich.

Ich überprüfe zuerst, ob alle System Variablen korrekt gesetzt sind:

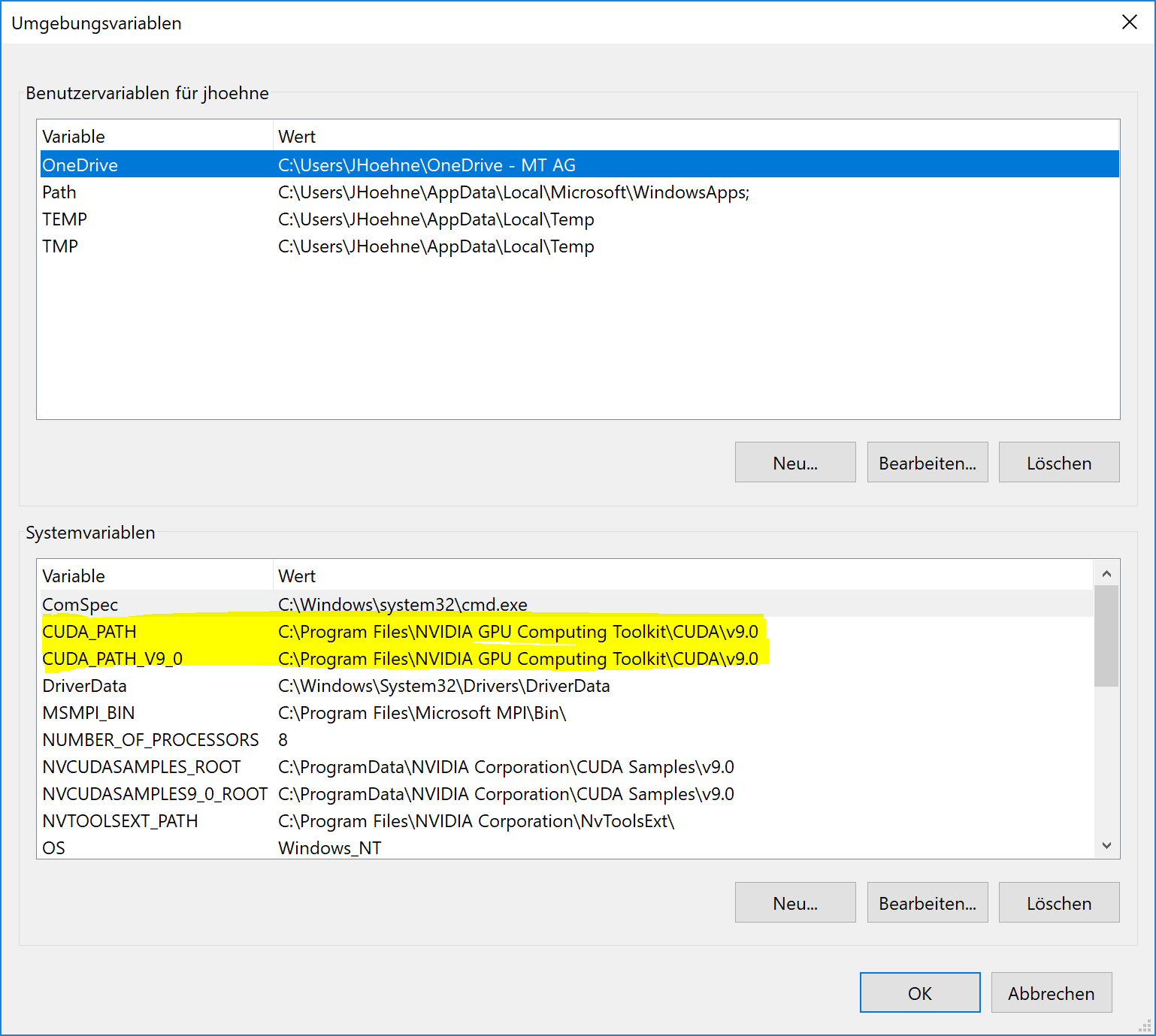


Abb 16: Die Systemvariablen CUDA\_PATH und CUDA\_PATH\_V9\_0 werden normalerweise automatisch gesetzt.

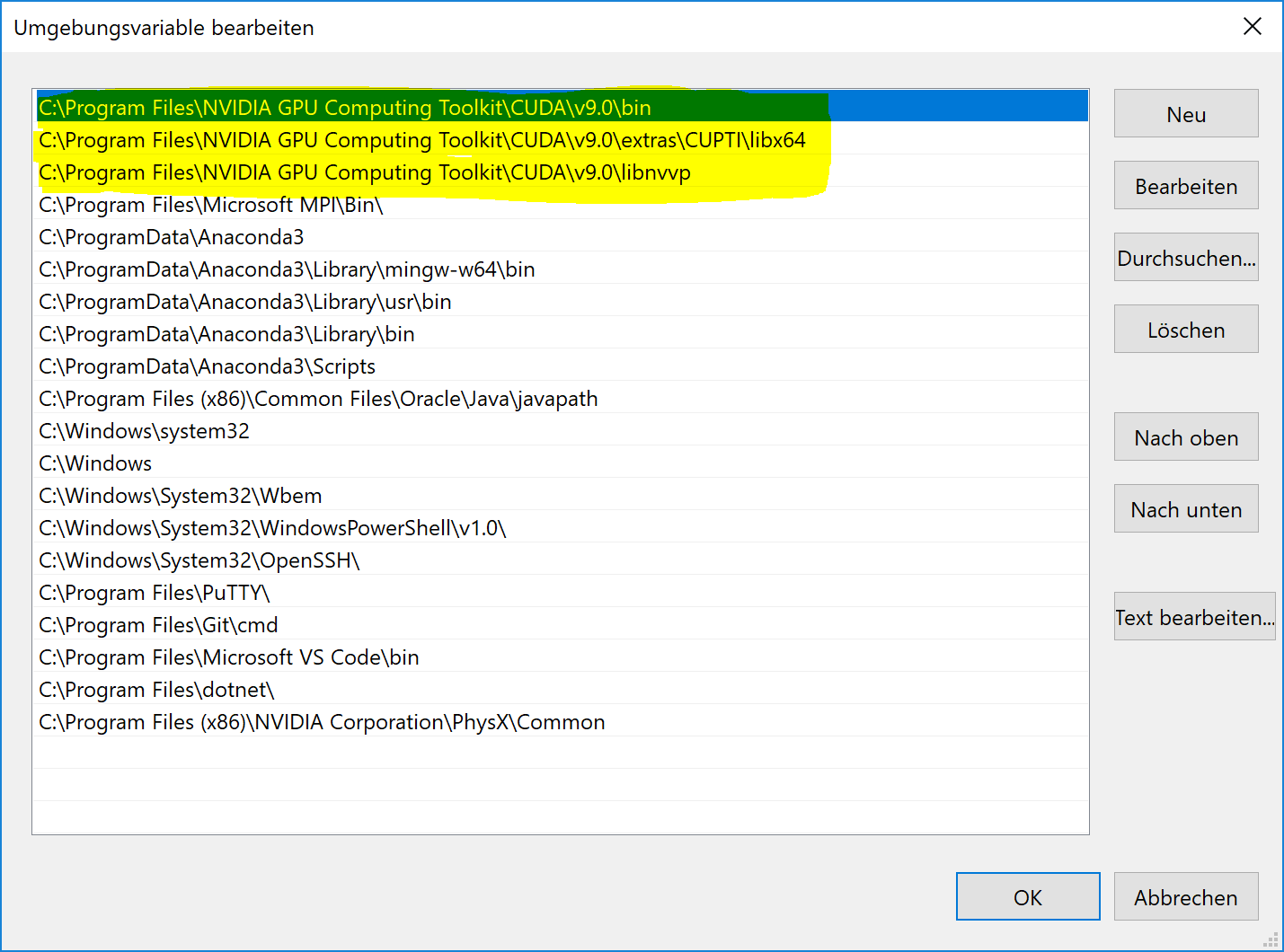
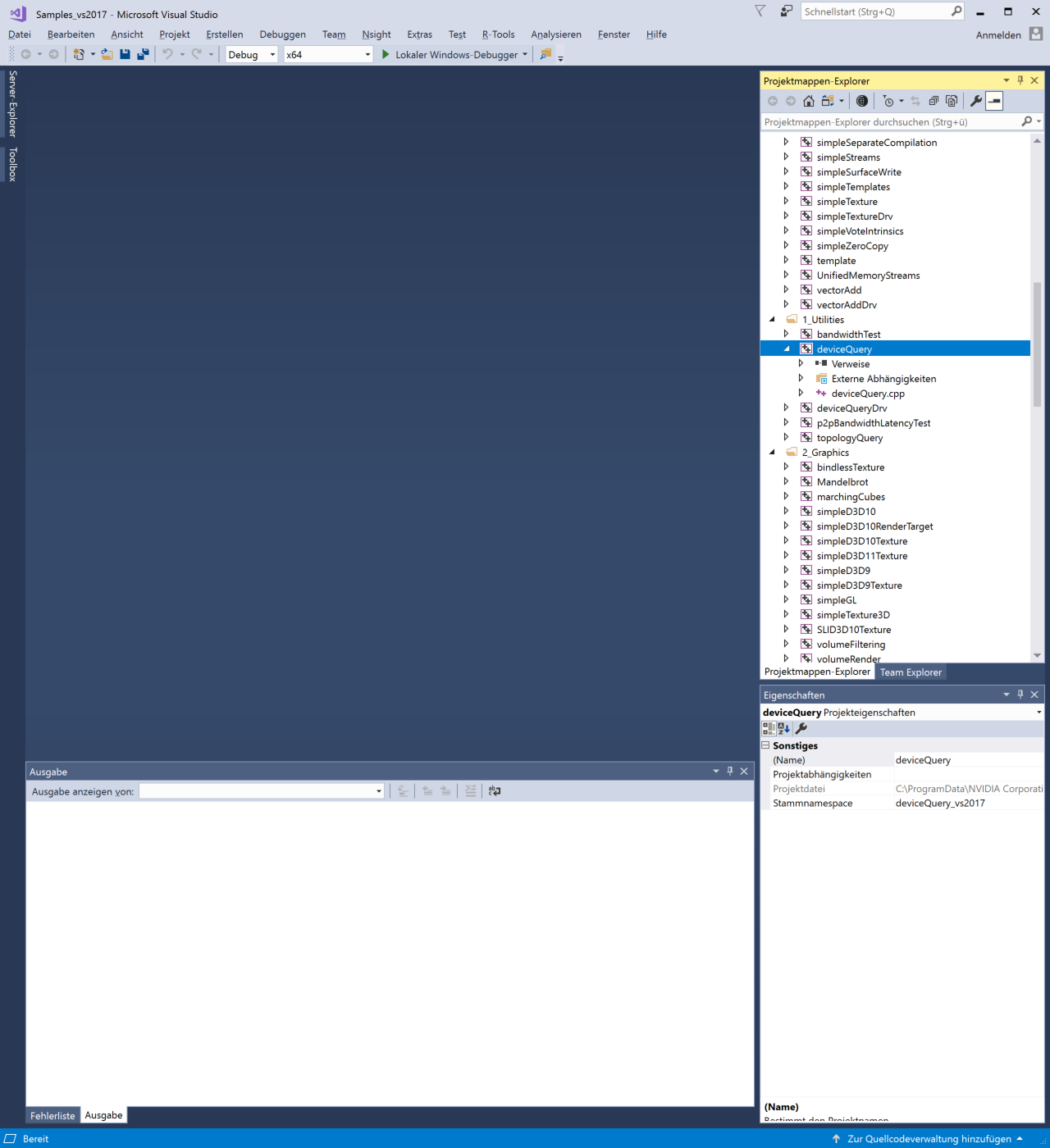
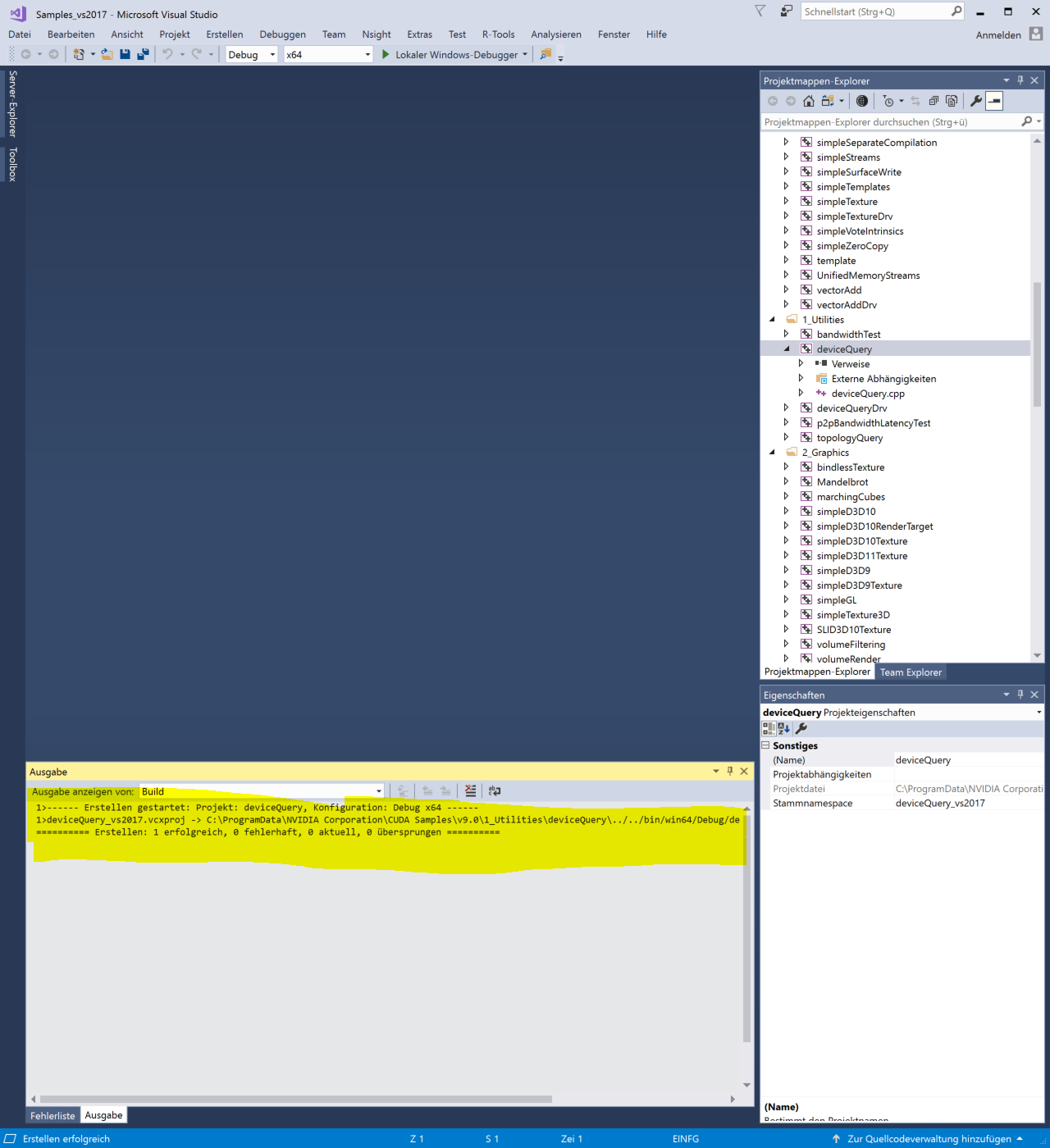
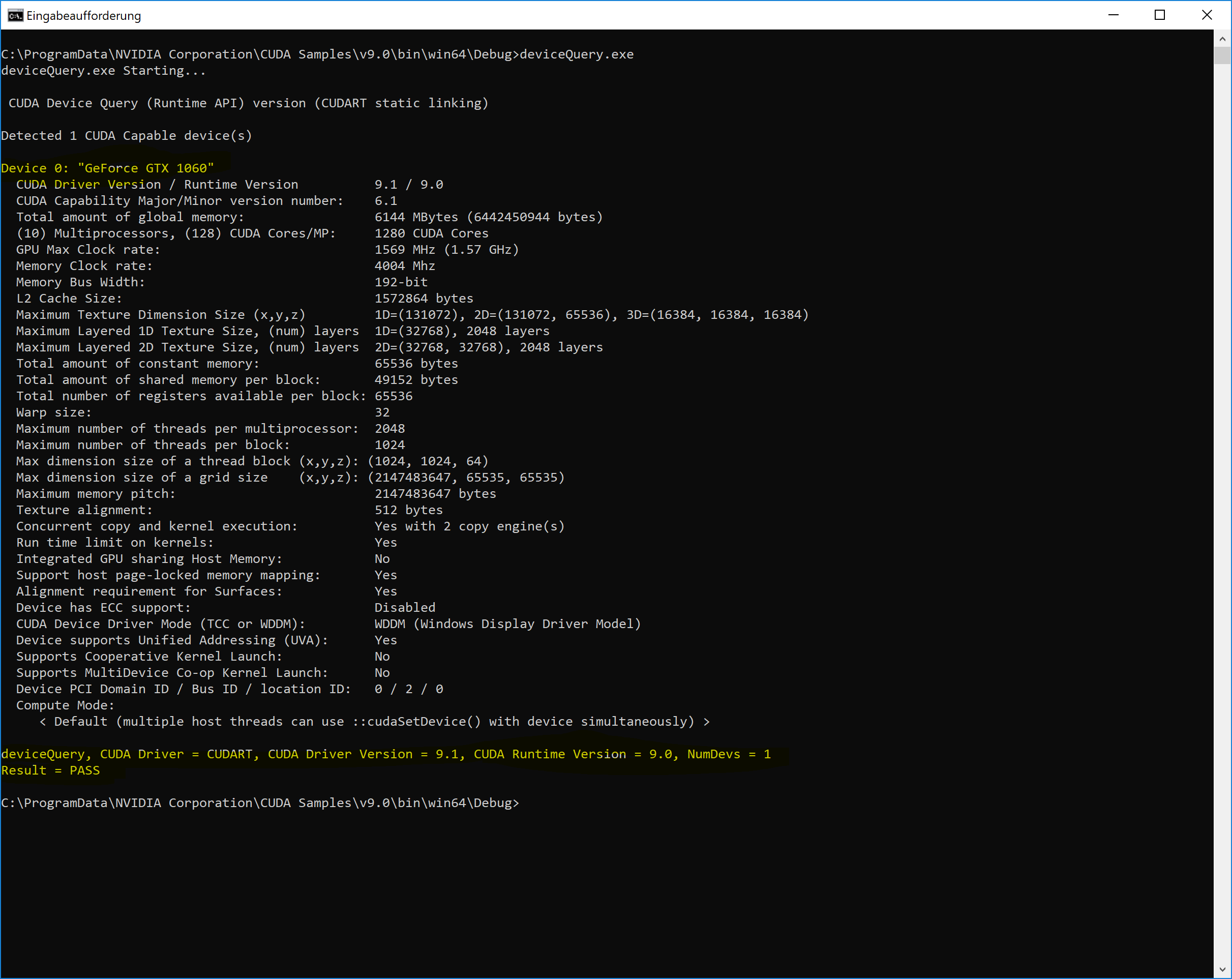


Abb 17: Im Windows PATH sollten die drei CUDA Einträge automatisch gesetzt worden sein.

Ich kompiliere den CUDA-Beispielcode, um zu überprüfen, ob CUDA ordnungsgemäß installiert wurde. Dazu starte ich Visual Studio und öffne das entsprechende Projekt. Die Sample-Code Projekte sollten hier abgelegt sein.: “**C:\ProgramData\NVIDIA Corporation\CUDA Samples\v9.0**”

1. Über den Menüpunkt **Date -> Öffnen -> Projek/Projektmappe...** lässt sich das Projekt **Samples\_vs2017.sln**” öffnen  
     
   Abb 18: Hier die Samples für Visual Studio 2017 auswählen
2. Nachdem die Projektmappe geöffnet wurde, selektiere ich im “Projektmappen-Exporer das Projekt “**deviceQuery**”   
     
   Abb 19: Ich habe nur die „DeviceQuery“ selektiert und compiliert
3. Nun wähle ich mit einem Rechtsklick im Kontextmenü „**Erstellen**“ um das Projekt zu bauen. Die Ausgabe sieht so aus:  
     
   Abb 20: Wichtig ist hier die „Erfolgsmeldung“ in der Ausgabe
4. Danach beende ich Visual Studio und starte meine **cmd**
5. Das Programm wurde nach **C:\ProgramData\NVIDIA Corporation\CUDA Samples\v9.0\bin\win64\Debug**” kompiliert.   
   Somit: “**cd** **C:\ProgramData\NVIDIA Corporation\CUDA Samples\v9.0\bin\win64\Debug**”
6. Ich starte “**deviceQuery.exe**”. Die Ausgabe sollte in etwa so aussehen. Wichtig ist hierbei eigentlich nur, dass die GPU unter Device 0 angesprochen wurde  
     
   Abb 21: Die Applikation deviceQuery.exe sagt Dir in der letzten Zeile, ob CUDA läuft oder nicht
7. Lehn Dich zurück und genieße den Anblick!

## Installation von Keras and TensorFlow

Die letzten Schritte sind nicht mehr schwer. Wenn die deviceQuery.exe von oben ausgeführt werden kann, ist es jetzt nur noch eine Frage der richtigen Versionen. Also

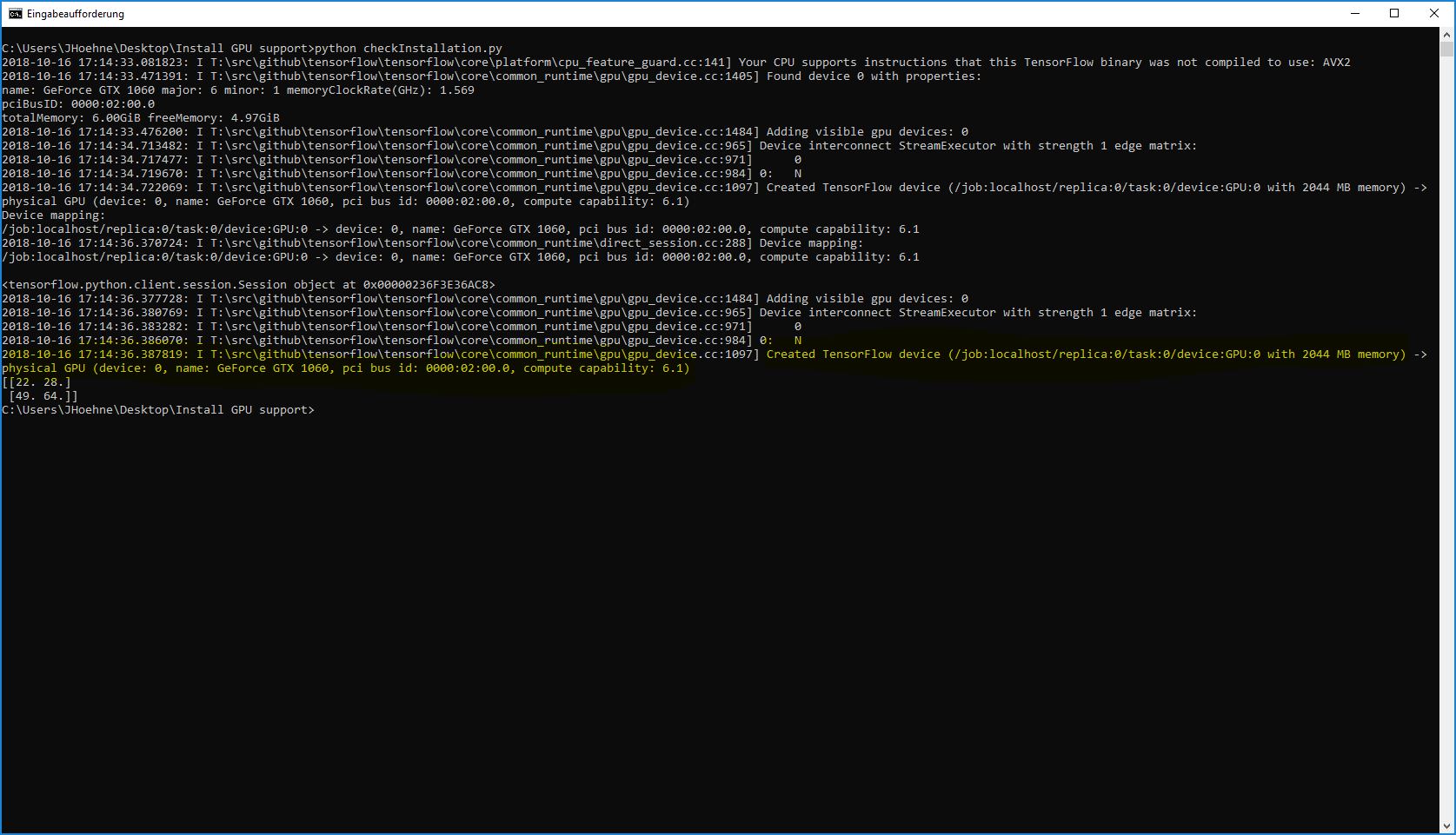
1. Starte die **cmd** als Administrator
2. Installiere TensorFlow 1.10.0.  
   “**pip install tensorflow==1.10.0**”
3. Installiere TensorFlow-gpu 1.10.0  
   “**pip install tensorflow-gpu==1.10.0**”
4. Installiere Keras 2.2.4  
   “**pip install keras==2.2.4**”

Ich habe mir folgendes Skript heruntergeladen. Eigentlich handelt es sich dabei nur um eine Art Hello World für TensorFlow-GPU (Achtung: Man sollte sich vorher überzeugen, dass es wirklich harmlos ist und keinen Schaden anrichten kann.)

<https://github.com/JJDoneAway/LearnPython/raw/master/GPU_Tweaks/CheckGU.py>

Ich habe das Skript lokal bei mir gespeichert und lasse es nun laufen:

1. Öffne deine **cmd**
2. Wechsle in das Verzeichnis mit dem CheckGPU.py Skript
3. **“python CheckGPU.py”**
4. Die Ausgabe sollte in etwa wie in Abbildung… aussehen. Wichtig sind die gelb markierten Zeilen, die zeigen, das die GPU angesprochen wurde.

  
Abb 22. Ein wenig versteckt wird Dir gesagt, ob TensorFlow auf der GPU oder CPU läuft.

Da das im Screenshot ein wenig klein ist, habe ich den entscheidenden Text hier nochmal herauskopiert: “**Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 2044 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1060, pci bus id: 0000:02:00.0, compute capability: 6.1)**”

Es muss GPU und nicht CPU sein

## Keras Code auf der GPU ausführen.

Wenn ich jetzt Keras Code laufen lasse, kann es sein, dass der folgende Fehler auftaucht:

**InternalError**: Blas GEMM launch failed : a.shape=(128, 784), b.shape=(784, 484), m=128, n=484, k=784

[[Node: dense\_1/MatMul = MatMul[T=DT\_FLOAT, \_class=["loc:@training/RMSprop/gradients/dense\_1/MatMul\_grad/MatMul\_1"], transpose\_a=false, transpose\_b=false, \_device="/job:localhost/replica:0/task:0/device:GPU:0"](\_arg\_dense\_1\_input\_0\_0/\_37, dense\_1/kernel/read)]]

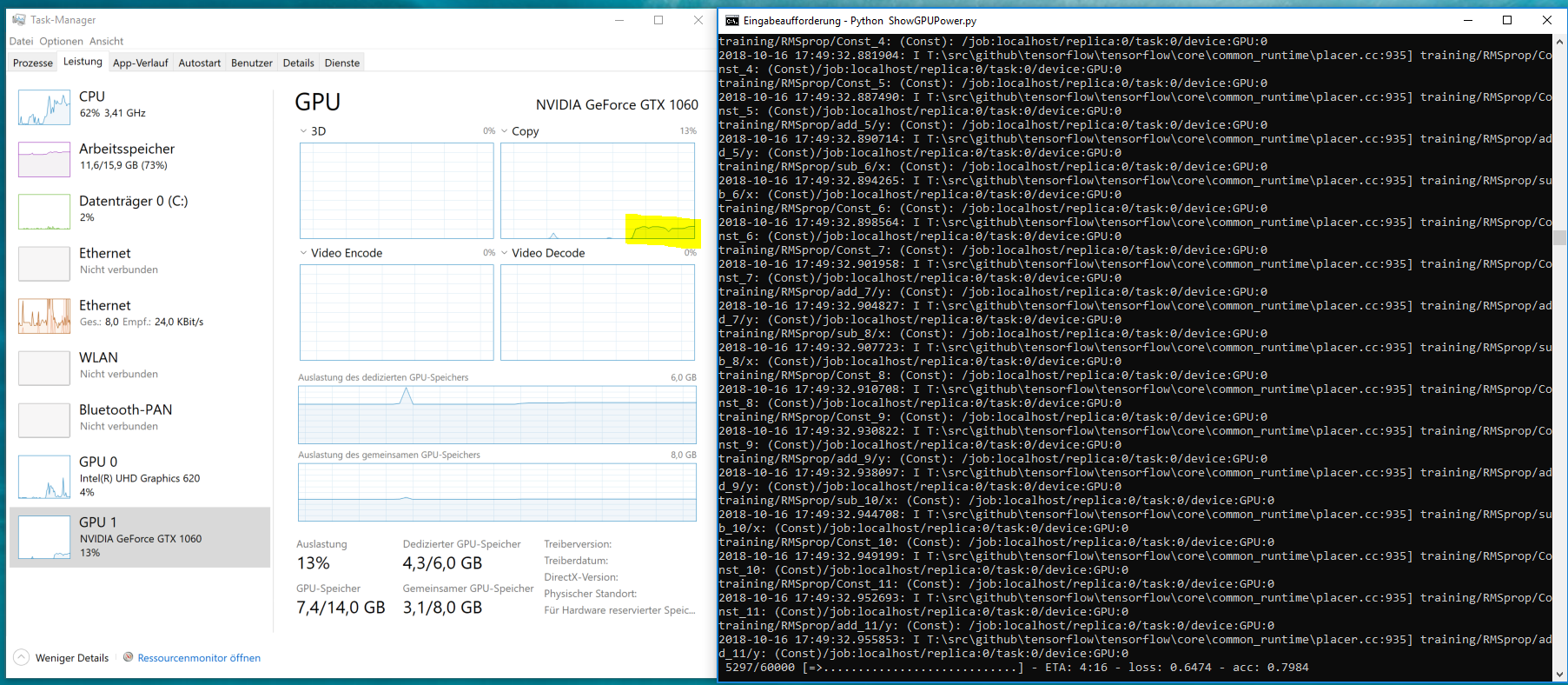
Um das zu vermeiden, musst man immer folgende Konfiguration in den Code kopieren:

<https://github.com/JJDoneAway/LearnPython/raw/master/GPU_Tweaks/KerasConfiguration.py>

Wenn man jetzt noch wirklich sicher gehen will, dass der Code auf der GPU läuft, kann man folgendes Python Skript laufen lassen. Dabei handelt es sich um ein Keras Model, das mit 500 Epochen gebaut werden soll. Natürlich macht das fachlich keinen Sinn, es verschafft aber die Zeit, während es läuft, den Task Manager zu öffnen und die die Auslastung der GPU anzusehen.

<https://github.com/JJDoneAway/LearnPython/raw/master/GPU_Tweaks/ShowGPUPower.py>

“**python ShowGPUPower.py**”

  
Abb 23: Die GPU 1 wird belastet, solange Du das Model trainierst.

Wenn Du das Skript mit „Strg“ + „C“ stoppst, wirst Du sehen, dass die Last sofort auf 0 fällt.

## Abschliessende Bemerkung

Es wäre schöner, wenn es eine einfache Install.exe gäbe, die alles installiert und uns nicht durch dieses finstere Tal der verschiedenen Versionen hetzen würde. Nichtsdestotrotz funktioniert es nun und es ist ja die Menge an Fehlern, die man in seinem Leben schon gemacht oder gesehen hat, die einen zu Senior werden lassen 😊

# Viel Spaß mit Deinen DNNs

# Kategorieen

KI, Python, Hardware

# SCHLAGWORTE

TenosorFlow, Keras, TensorFlow-gpu, GPU, CUDA, cuDNN, Windows 10, NVIDIA, install GPU support, TensorFlow 1.10.0, SurfaceBook 2

# ZUSAMMENFASSUNG

Installiere die NVIDA CUDA GPU-Unterstützung unter Windows 10 für TensorFlow 1.10.0 und Keras 2.2.4.