Building an OS Image for Deep Learning

Daniel Gunzinger, Benedikt Jaeger*, Sebastian Gallenmüller*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: daniel.gunzinger@tum.de, jaeger@net.in.tum.de, gallenmu@net.in.tum.de

***Abstract*—This paper documents the process of creating an reproducible OS image for deep learning. The target framework is Tensorflow 2, which is provided in a recent version in this image, accompanied by support for GPU acceleration, which can improve performance and thus development time significantly. During the creation of this OS image, a number of problems have been encountered and solved. Due to the created OS image, it is now possible to rapidly spin up a server, which can immediately provide a reproducible environment fit for development and training of deep learning applications that utilize the Tensorflow 2 framework with GPU acceleration.**

***Index Terms*—operating system, tensorflow, gpu, deep learning, cuda, cudnn**

1. Introduction

In this paper the process of building an OS Image based on Debian Bullseye in order to provide GPU support for deep learning applications using Tensorflow 2 with the new generation of Nvidia RTX Super graphics cards is discussed.

This aims to provide a reliable and reproducible OS image for the development and training of deep learning applications, reducing development time by removing tedious setup tasks and improving execution performance of the developed applications with GPU acceleration.

The OS image also aids the reliability of testing results of any executed applications as the exact configuration of the whole OS image can be preserved and tests can be repeated by loading the OS image again on the same target network node. Thereby providing the same hardware and software environment and thus a reproducible platform that can help in achieving reliable application testing results for all users of the OS image.

This paper is structured as follows, in Section 2 background information about the used software and hardware is provided, including the infrastructure setup which the OS image is targeted for. In Section 3 the building process of the OS image is detailed, this contians information about the testing and building methodology and the problems that were encountered during this process. Section 4 provides an overview of the performance testing that was done including benchmark results for the used test scripts on a target server and its hardware specifications.

1.1. Deep Learning

Deep Learning is a type of machine learning which can assign a given input to a set of known elements based on previous training.

Deep Learning is often used to classify images or recognize known elements in a given input image, but it is not constrained to only work on images, but can also work on other input data formats such as text or sound as DeepL[1] or speech recognition software show.

However in this work, the focus is to provide GPU acceleration of deep learning frameworks primarily concerned with image recognition.

For this purpose it is required to create a model with a number of different layers which process the input image and pass their output on into the next layer.

There are different layer types which perform different computations on the input they recieve and also differ in their output and output formats.

One layer type which is commonly used in image recognition models are convolutional layers, this layer type performs operations on a given range of the input matrix. In this type of layer each output is the result of a function over a window of the input.

Another commonly used layer type are fully connected layers, where each input is connected to each output by a weight. In this type of layer each output is the result of an operation involving every single input and its weight.

One more important layer type are pooling layers which are used to filter the input, perform a specific operation on a range of inputs and combine them into one output.

2. Background

In this section the targeted infrastructure setup is detailed, and choices for software and hardware are explained.

2.1. Infrastructure Setup

The targeted infrastructure setup consists of a server network which is managed through the `pos` software, which is an abbreviation for plain orchestrating service.

Among many other features out of scope regarding this paper, it allows for the use of a calendar. In this calendar multiple users working with the server network can enter dates, durations and targeted servers in order to reserve them for their later use, and allows for an overview of the current reservation status of the available servers. When a user of this system has a ongoing reservation of one or multiple servers, it can be used to allocate the specified

1. https://www.deepl.com/press.html

doi: 10.2313/NET-2020-04-1_05

servers. Once one or multiple servers are allocated by a user, they can be started, stopped and reset individually. Available OS images can be chosen for deployment and an overview of their operational status can be displayed.

For the image building process the `mandelstamm` software is used to create OS images that can be used in the pos software mentioned above. It consists of a set of build scripts which are used to specify the contents of the OS image, such as the base operating system and software that is supposed to be installed once the image is executed, and to configure the image for deployment.

The `pos` and `mandelstamm` software are projects by the Chair of Network Architectures and Services and are not publicly available.

2.2. Software

In this section the software components of the OS image are described.

As we need a reliable operating system, Debian was chosen since it provides a stable base system with a broad range of software packages being available for installation and use through the repositories. In particular Debian Bullseye (11), the current testing testing release has been chosen.

As the main focus of this work is to provide an OS image for GPU accelerated deep learning, deep learning libraries need to be available too. Thus Tensorflow 2 and PyTorch are installed through the `pip` package manager and included in the OS image.

While Tensorflow 2 provides APIs in multiple different programming languages[2], we focused on providing support for a Python workflow, which is also required for PyTorch.

Python is available in many different versions, with incompatibilities between the major versions 2 and 3, for our OS image we aimed to provide a recent version of Python 3.7, which is available through the Debian repositories and is also supported for use with the Tensorflow 2 library.

As a major task of this work is to provide GPU acceleration for Tensorflow and PyTorch, the GPU driver and multiple libraries from Nvidia also need to be included into the OS image.

The installed version of the drivers is 430.64, with CUDA 10.1 being provided by `libcuda1` and cuDNN 7.6.5.32 installed through Nvidias generic package for linux targets. Other important libraries in order to provide GPU acceleration are `libcudart10.1`, `libcublas`, `libcufft`, `libcurand`, `libcusolver`, `libcusparse` and the `nvidia-cuda-toolkit`.

2.3. Hardware

The decision to work with the Nvidia graphic cards stems from their hardware acceleration capabilites for compute and deep learning applications.

The amount of streaming processors present on these cards is useable through the CUDA API which can provide impressive speedups for Tensorflow programs over execution on general purpose CPUs. Another feature which the Nvidia RTX series GPUs provide are the Tensor Cores, which can provide another speedup over common general purpose hardware for mixed precision matrix multiplications, commonly used in the network training phase.

3. Building Process

In order to build the images the `mandelstamm` software is used to create and package the OS images.

For the first attempt at creating the target OS image Debian Buster was chosen as the operating system, as it is the latest stable release version. In order to enable support for the used GPUs the `nvidia-driver` and the `nvidia-smi` (system management interface) were included into the OS image.

When attempting to test the created image on the targeted server, it became apparant that the version of the `nvidia-driver` package available in Debian Buster is not recent enough to support the used RTX 2080 Super GPUs, as elaborated upon in Section 3.2.1.

Thus the built image was tested on a server containing two Nvidia GTX 1080 Ti GPUs in order to determine if the approach for driver installation had succeeded. The installation success could be confirmed by executing the `nvidia-smi` utility which reported the two GPUs with their correct names and further statistics such as current power consumption and memory usage.

The next step was to install Tensorflow 2, which instead of building it from source, can be acquired via the Python `pip` utility. During the installation of Tensorflow 2 the latest version available through `pip` was used, at the time of this testing this was version 2.0.0, which was released on September 30th of 2019[3].

This installation led to the discovery of the next problem due to Tensorflow 2 requiring at least version 19 of the `pip` utility, which is not provided in any of the Debian repositories as described in Section 3.2.2.

This required the `pip` utility to not be installed through the `apt` package manager using the Debian repositories, but instead through downloading a provided installation file from the Python Package Authority (PyPA) and executing it in order to complete the installation[4].

Thereafter the installation of Tensorflow 2 was again attempted by installing version 2.0.0 through the `pip` utility, this time completing successfully and thus enabling first tests to be run.

Since the goal of this image is to provide GPU acceleration support with the Tensorflow 2 library, the first test was to see if the GPU is recognized as a processing device by Tensorflow 2 as described in Section 3.1.1.

This revealed warnings about a number of additional libraries needed in order to register the GPU successfully, and `libcudnn`.

All of these libraries except for `libcudnn` are available through the Debian repositories, however installing them was of no help since the available versions did not match the required CUDA version, as described in Section 3.2.3.

At this point the upcoming Bullseye release of Debian, version 11, has been chosen due to the package availability problems and a lack of driver support for the targeted

2. https://www.tensorflow.org/api_docs

3. https://pypi.org/project/tensorflow/#history

4. https://pip.pypa.io/en/stable/installing/

doi: 10.2313/NET-2020-04-1_05

graphics card series that were encountered with Debian Buster.

As the Bullseye release of Debian is still in development, neither the official release date nor the end of life date is known yet, however extrapolating from the Debian version history[5] a release date in 2021 and an end of life date around 2024 would match the current pace of releases.

By changing the target OS to Debian Bullseye a problem with the `mandelstamm` build scripts became apparent, as the URL format for the security repositories had changed for Debian Bullseye as elaborated upon in Section 3.2.4. Thus the `mandelstamm` build scripts had to be adapted in order to successfully build the OS image.

After this problem was addressed, the build script for Debian Bullseye was modified by adding calls to the `apt` package manager in order to install the Nvidia GPU driver and system management interface from the Debian repositories. Another call to the package manager was added in order to install the aforementioned additional GPU libraries which are fortunately available in the required and matching version in the Debian Bullseye repositories.

Afterwards the built Debian Bullseye image was deployed to the test server with the Nvidia RTX 2080 Super GPU, and the `nvidia-smi` command was called in order to determine correct installation of the GPU drivers. This could be confirmed as the `nvidia-smi` utility did now provide the correct name of the card in the output alongside the other statistics, it also reported both the `nvidia-smi` and driver version as 430.64, which officially supports the targeted card as listed in the release notes for this driver[6].

After all necessary tools and libraries have been successfully installed Tensorflow 2 can be installed via `pip`, however the version that is going to be installed has to match the installed CUDA and cuDNN version. In our case the chosen version was version `2.1.0rc0`, which is the first release candidate of Tensorflow version 2.1 and requires CUDA 10.1 and cuDNN >=7.4[7].

After the first confirmation of driver support for the installed card the next testing stage was executed by listing available devices in Tensorflow as described in more detail in Section 3.1.1. This returned the expected result of a processing device list with one CPU and one GPU device being available.

By running the test script in order to verify the availability of GPU acceleration a problem with Tensorflows memory allocation behaviour on GPUs became apparent, which is described in greater detail in Section 3.2.6.

After solving the GPU memory allocation issue, all parts of the test script could be executed successfully on both CPU and GPU, demonstrating full functionality of the OS image.

3.1. Testing and Deployment

This section explains the methods used for testing the functionality of the produced OS images.

5. https://wiki.debian.org/DebianReleases
6. https://www.nvidia.com/Download/driverResults.aspx/153714/en-us
7. https://www.tensorflow.org/install/source#gpu

3.1.1. Initial testing by listing devices. For initial testing the following two lines of code were used to list the available processing devices, such as the CPU and GPU.

```
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

3.1.2. Deep learning test script. In order to test the functionality of the Tensorflow installation and to ensure that the installed CUDA and cuDNN versions work with the chosen version of Tensorflow a custom test script has been created.

It trains a neural network for image classification using the CIFAR10 dataset[8], and is structurally similar to a Tensorflow tutorial example for CNNs[9].

The script has two major sections, in the first section convolutional layers are used, this section can be disabled and thus skipped. This first section contains two Conv2D layers[10] with a Max-Pooling layer in between the two Conv2D layers.

The second section of the script uses fully connected layers, thus the input is first flattened. After the input has been flattened two fully connected layers are added as Dense layers[11], with the first layer using a ReLU activation function and the second (final) layer using the softmax activation function.

Using these layers as described above, the model is then trained for ten epochs over all images contained in the CIFAR10 dataset.

3.2. Encountered Problems

This section elaborates on the problems that were encountered during the creation and testing of the OS images and their solutions.

3.2.1. Target GPU not supported in Debian Buster. Due to the first build targeting Debian Buster, the latest version of the driver available in the Buster-specific repositories was installed, which was version 418.74.

However as we need to support an Nvidia RTX 2080 Super GPU this is not recent enough, as it does not support any of the Super-series cards, which were released in July of 2019[12].

This was noticed due to the nvidia-smi utility output not reporting the name of the installed GPU correctly.

3.2.2. Tensorflow software requirements. There are also problems with the availability of recent Python3 `pip` versions on several Debian versions including Buster and Bullseye, as the repositories only provide `pip` version 18.1, yet at least version 19 is required for our target application Tensorflow 2.

8. https://www.tensorflow.org/datasets/catalog/cifar10
9. https://www.tensorflow.org/tutorials/images/cnn
10. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D
11. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense
12. https://www.anandtech.com/show/14663/the-nvidia-geforce-rtx-2080-super-review

doi: 10.2313/NET-2020-04-1_05

3.2.3. Mismatched GPU library versions in Debian Buster. While the repositories for Debian Buster do contain libraries for CUDA 10.1 support, none of the other important libraries for GPU acceleration support are available in version 10, instead only in version 9.2, which could not be used successfully in combination with Tensorflow 2. The problematic libraries are `libcudart`, `libcublas`, `libcufft`, `libcurand`, `libcusolver` and `libcusparse`.

3.2.4. Missing support for Debian Bullseye in mandelstamm. As `mandelstamm` does not have a specific build script for Debian Bullseye, an image creation was first attempted by copying the build script for Debian Buster and changing the release name from Buster to Bullseye.

This however did not result in a successful image creation as the security repositories could not be found. After closer inspection of the build scripts and Debian documentation, an adjustment had to be made to the generic Debian build script as the URL of the security repository had its format changed[13], thereby creating a special case in the build script.

3.2.5. Installation of the cuDNN Package. An important library regarding GPU acceleration for deep neural networks is Nvidias cuDNN package. It is only available through a repository for Ubuntu, installations on other distributions require a more generic package available for download on Nvidias website through the Nvidia Developer Program[14], which however requires a membership in the mentioned developer program.

Thus it is necessary to install the package manually according to Nvidias documentation[15].

3.2.6. Issues with cuDNN and Tensorflows default settings. When executing the test script on a GPU device, an error about not being able to allocate memory was returned. This turned out to be a configuration issue instead of a driver or library issue and has been solved by adding a small loop which iterates over the available GPU devices and calls the following function for each GPU device:

```
tf.config.experimental.set_memory_growth(device,
True)
```

After setting this flag for each GPU device the convolutional network part of the test script could be run without issues on the GPU device, which as described in Chapter 4 allowed for a significant speedup over executing it on the CPU.

3.2.7. Default build options of available Tensorflow packages. During the execution of the test scripts on the CPU using the Tensorflow `2.1.0rc0` build obtained via the `pip` package manager the following warning was logged:

```
Your CPU supports instructions that this
TensorFlow binary was not compiled to use:
AVX2 AVX512F FMA
```

Which implies that the execution times observed using when the CPU for the test script could be significantly lowered by building Tensorflow from source with enabled support for the advanced AVX instruction sets and the fused multiply-add instuction set, which both can accelerate a common operations of deep learning applications significantly.

This is a problematic default build setting as a majority of recent CPU architectures, starting with Intels Haswell microarchitecture in June 2013[16] include support for the AVX2 and FMA instruction sets.

If support for using these instruction sets is added in future builds a meaningful speedup could be observed when running deep learning applications without GPU acceleration, as the SIMD instruction sets AVX2 and AVX512 can improve the throughput of matrix multiplications and other common operations in multiple neural network types.

4. Performance Evaluation

In this section we compare the performance of the test script when running on the CPU and the GPU.

The performance testing was conducted on a server with the relevant specifications listed in Table 1.

TABLE 1: Testing server specifications

Part	Equipped
Processor	Intel Xeon Silver 4214 (12c24t, up to 3.20GHz)
Memory	8x32GiB DDR4 2400MHz in hexa-channel
Mainboard	Supermicro X11SPi-TF
Graphics card	Nvidia RTX 2080 Super

The test script (see Section 3.1.2) was used and the complete execution time for each configuration was obtained with the time command.

In order to show the speedup of different operations, two different configurations were used for the test script execution. First, the complete script including all operations was executed. For the second configuration the test script was modified to skip the execution of convolutional operations.

Both of these configurations were executed three times with and without GPU acceleration and the execution time for these tests was then averaged in order to alleviate the effects of run-to-run variance.

TABLE 2: Performance testing results

Test	CPU execution time	GPU execution time
Complete	763.65 (14945.69)	90.73 (156.78)
Fully connected network	107.00 (1439.07)	63.49 (126.29)

Results format: real time (user time) in seconds

The results listed in Table 2 show that GPU acceleration provides a significant speedup, especially when working with convolutional networks, which can be explained by inspecting the functionality of the layers and the capabilities of the used hardware.

13. https://www.debian.org/releases/bullseye/errata#security
14. https://developer.nvidia.com/rdp/cudnn-download
15. https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html
16. https://en.wikichip.org/wiki/intel/microarchitectures/haswell_(client)#New_instructions

doi: 10.2313/NET-2020-04-1_05

Some of the operations in the script described in Section 3.1.2 are rather compute bound, while other types of operations, such as pooling or the processing of activation functions is rather memory bandwidth bound[17]. The parallelizability of the layers is an important aspect of the observed performance scaling, as the amount of compute cores differs greatly between the used CPU, where 12 cores with 24 threads are available, and the used GPU, where 3072 shader processors and 384 tensor cores are ready to compute.

In memory bound operations the GPU will also have an advantage as it features a 256bit wide memory bus operating at 15500MT/s resulting in a theoretical peak bandwidth of 496GB/s to the GDDR6 chips. In comparison the CPU can access six channels of DDR4 memory with a width of 64bits each, resulting in a 384bit wide memory bus operating at 2400MT/s resulting in a theoretical peak bandwidth of 115.2GB/s.

With a complete execution of the test script we can observe a speedup of 88.1% when enabling the GPU acceleration over a CPU only execution. By disabling the execution of the convolutional layers, the difference in execution time shrinks significantly, however enabling GPU acceleration still yields a 40.7% decrease in runtime.

With these numbers we can also see that the execution of convolutional networks profits much more from GPU acceleration, as the execution time compared to the reduced test configuration increases by 42.9% while the time taken when executing on the CPU increases by 613.7%.

However, it is important to note that the chosen release for Tensorflow 2 (`2.1.0rc0`), seems to lack support for CPU instruction sets that could improve the execution time when running on the CPU significantly, as described in Section 3.2.7.

5. Conclusion and Future Work

The created OS image supports the use of GPU acceleration with Tensorflow 2, which provides a significant reduction in runtime for deep learning applications, especially in applications which include convolutional neural networks.

Currently the OS image is based on Debian Bullseye for the operating system, featuring the Nvidia drivers in version 430.64 for support of their latest series of graphics cards, as well as a number of accompanying libraries.

Other installed software includes Python 3.7, a recent version of the `pip` package manager and most importantly recent versions of Tensorflow 2 and PyTorch.

With the OS image ready for deployment, time is saved in the development workflow as tedious setup tasks can be skipped by deploying the OS image to an available server and using it for the development tasks.

The building process did also show that in order to create an environment featuring recent versions of the Tensorflow and PyTorch frameworks with GPU acceleration support, special attention needs to be brought to the used graphics cards and the GPU driver version, as well as the available libraries regarding GPU acceleration, as these libraries have dependencies on specific CUDA versions.

For the future the OS image can be extended to include more available deep learning frameworks as well as more software tools that ease the development workflow.

Additional images using other operating systems, e.g. Ubuntu as a basis could also be created in order to expand the available software support through more repositories, thus allowing for more use cases and letting developers choose an environment with which they are already familiar.

If a version of this image is to be created for CPU only execution of Tensorflow 2 applications, it would be beneficial to check the target CPU for its supported instruction sets, compile Tensorflow from the source code. By including all instruction sets that the target CPU can support in the compilation settings a performance advantage over the precompiled binaries available through `pip` can be achieved.

17. https://docs.nvidia.com/deeplearning/sdk/dl-performance-guide/index.html

doi: 10.2313/NET-2020-04-1_05