Paper PP10

Preparing Big Data for Analysis

Thomas Ellebæk, Ferring Pharmaceuticals A/S, Copenhagen, Denmark

ABSTRACT

Training a deep neural network on terabytes of e.g. imaging data requires both computing power and proper parallelization techniques. When the amount of data exceeds what can be stored in memory, batches of data needs to be read repeatedly from more persistent storage. This poster will explore the TensorFlow™ data API (tf.data), the Keras Sequence class and HDF5 files in support of training deep neural networks. Performance will be evaluated using a publicly available benchmark dataset. Based on this evaluation suggestions will be made towards how to organize and store image data for analysis using Python, the TensorFlow™ framework and the Keras API.

INTRODUCTION

At Ferring Global Biometrics we are planning to train a deep neural network on a large amount of image data (+500 GB) using TensorFlow™ or Keras via Python. The data will be available to Ferring from start of 2020, so we haven't started the actual modelling yet. However, in order to train a model on a large amount of data we need an efficient way of preparing and pipelining an image repository in support of deep learning. We have explored three different approaches; The TensorFlow™ data API (tf.data), the Keras Sequence class and HDF5 files.

For the experiments covered in this poster we have used the publicly available dataset Caltech256, which is a collection of 30,607 images (1.2 GB) in 256 categories and one additional category of background images grouped under the name 'clutter'.

The Ferring Global Biometrics image repositories reside in Azure in either file shares or blob storage.

LOADING IMAGES IN PYTHON

To compare methods for pipelining and preprocessing of image data in support of deep learning, we need to be able to load images from the image repository and get hold of the pixel intensities of the available color channels. There are several ways to do this in Python. We have explored a set of the more popular Python packages supporting loading of images

- **OpenCV** (https://opencv.org/): Open Source Computer Vision Library.
- **PIL** (https://python-pillow.org/): The Python Image Library. Through pillow package.
- **imageio** (https://imageio.github.io/): Python library providing an easy interface to read and write a wide range of image data.
- **scikit-image** (https://scikit-image.org/): Image processing routines for SciPy - NumPy, Matplotlib, pandas. Versatile set of image processing routines. Working as API to other toolkits (specify plugin), default = imageio.

The results of the image loading experiment is presented in Figure 1. The performance using scikit-image package is not reported since it effectively became a wrapper for the underlying imageio methods. The results clearly show that loading blobs is faster than loading jpegs from file shares. The blob loading relies on the azure-storage-blob Python package in all three cases, so what the figure shows is that decoding the blob into a multidimensional array of pixel intensities is equally fast using the different Python packages. The API for working with the imageio functionality was found to be simpler than PIL and OpenCV, so imageio was selected for creation of HDF5 file and to be used in the sequence generator in the main experiment.

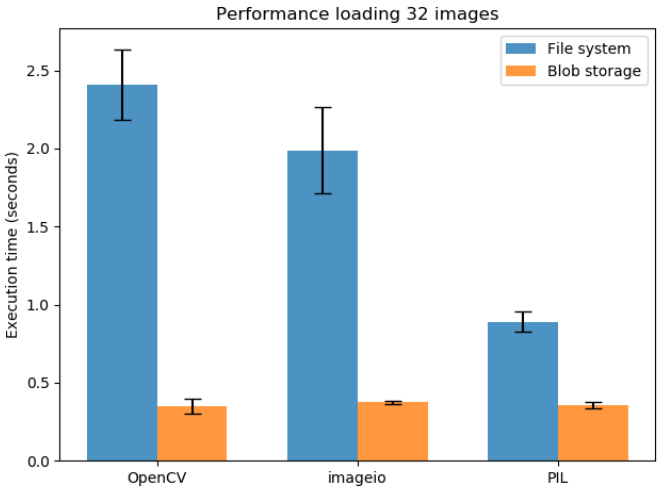

Figure 1: Comparing image load performance using three different Python packages. Reporting mean +/- standard deviation of 10 repetitions of loading 32 images.

THE MAIN EXPERIMENT

Three different approaches to pipelining batches of persistent data to Keras model training with TensorFlow™ backend has been explored in this poster.

- **tf.data**: TensorFlow™ data API. Iterating over dataset of file paths and repeatedly loading and preprocessing images via tf.io.read_file method. Not compatible with Azure Blob Storage. Batch shuffling not available.

- **Keras**: Custom Keras sequence generator to load and preprocess images batch by batch from Azure Blob Storage via imageio.imread method.

- **HDF5**: Custom Keras sequence generator to load preprocessed arrays of pixel intensities batch by batch from HDF5 file (Hierarchical Data Format). Not compatible with Azure Blob Storage. Notice that the time it takes to generate the HDF5 file from original images (via the imageio.imread method) is substantial, but not part of the evaluation since it is only done once.

In the experiment we are measuring performance as the time it takes to perform a task. We are reporting mean +/- standard deviation of 10 repetitions, and we are timing batch loading with and without subsequently model training. The model training involved in this experiment is transfer learning based on copies of the MobileNet v2 from tf.keras.applications. System configuration and Python packages used for the experiment is listed after the conclusion.

RESULTS

If your design allows you to read sequential batches of data without shuffling, it is worth exploring the tf.data API, as illustrated in Figure 2. Furthermore, if you have a smaller dataset, and shuffling of all records is preferred, tf.data also seems to be efficient. However, this paper is focusing on preparation of big data for analysis, and already at the size of 30,607 images, shuffling with tf.data becomes a burden. Probably because batch shuffling is not available with the TensorFlow™ data API.

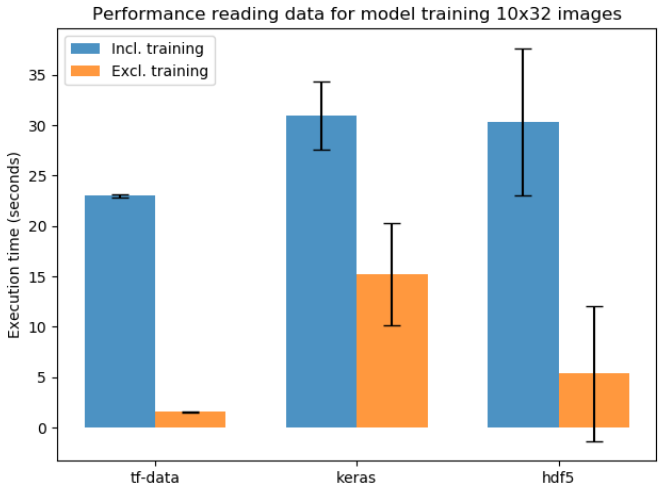

Figure 2: Comparing sequential batch loading without data shuffling, with and without model training. Reporting mean +/- standard deviation of 10 repetitions of loading 10 batches of 32 images.

Reading 10 random batches of 32 images from the Keras and HDF5 sequence generators, showed that on average blob loading and preprocessing is faster than reading preprocessed data stored in HDF5 file on file share.

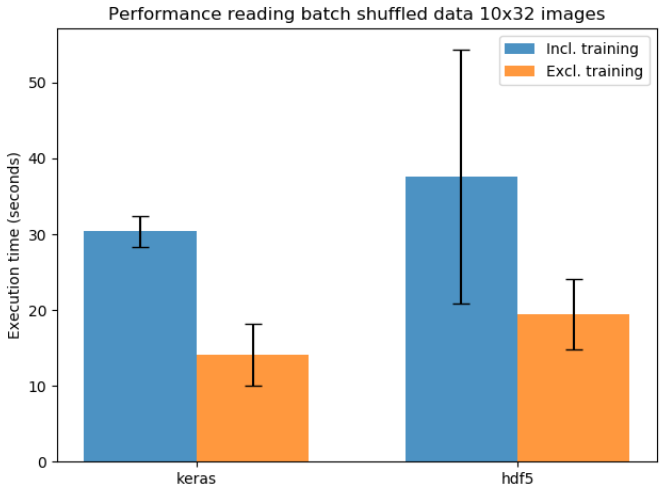

Figure 3: Comparing random batch loading (batch shuffling for each epoch of training), with and without model training. Reporting mean +/- standard deviation of 10 repetitions of loading 10 batches of 32 images.

CONCLUSION

Building an efficient data pipeline for deep learning model training remains a challenge where the optimal solution depends on various factors like experiment design, dataset size and hardware configuration.

The TensorFlow™ data API is most efficient for sequential batch reading without data shuffling. However, it is recommended to shuffle data between learning epochs, and with big datasets, the tf.data API do not provide an efficient way to shuffle data properly. The most efficient batch shuffling data generator was obtained as a custom Keras sequence generator reading images from blob storage and repeatedly preprocessing them in each epoch.

SYSTEM AND SETTINGS

The work in this poster was performed on an Azure Data Science Virtual Machine of type NC6s-v2, with 6 virtual CPUs, 1 NVIDIA Tesla P100 GPU and 112 GiB Ram. The experiments were performed in jupyter notebook with Python 3.7.4 via the open-source Anaconda Distribution. Following Python packages were installed.

Python Package	Version
numpy	1.16.5
tensorflow	1.14.0
keras	2.2.4
h5py	2.9.0
cv2 (OpenCV)	4.1.1
pillow	5.4.1
imageio	2.5.0
scikit-image	0.15.0
azure-storage-blob	2.1.0
matplotlib	3.1.1
notebook	6.0.1
conda	4.7.11
pip (to install opencv-python in Windows)	19.2.2

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Thomas Ellebæk
Ferring Pharmaceuticals A/S
Kay Fiskers Plads 11
2300 Copenhagen S
004528787983
Thomas.Ellebaek@ferring.com