Exercise 3

Edge Computing

Daniel May (11809922)
Julia Putz (51841360)
Philip Vonderlind (11739128)

Data-intensive Computing, SS2022

TU WIEN Informatics

June 28, 2022

Development

We decided to do the audio recognition task and followed the supplied tutorial for the base implementation. Audio files are converted to spectrograms, which show frequency changes over time. These spectrograms are then fed into a convolutional neural network, where the input is first downsampled with a resizing layer and normalized as preprocessing steps.

We decided to train the model beforehand and save it in our application's folder. This is closer to reality because a pre-trained model is loaded on the application startup.

Further, we used the reduced dataset (in size BIG), since we had problems uploading the entire dataset to the Hadoop cluster.

The dataset was uploaded entirely at once and at the request time, only the path to the file was provided. This made the development of the app easier and is allowed according to the tutors.

In the `Dockerfile`, we had to provide the path to the pre-trained model. Additionally, we switched to `Python` version `3.9.13` since with the existing version, we encountered some errors with `flask`.

In the `requirements.txt` we added the `pathlib` library for simpler filesystem handling, especially between Windows and Unix based systems.

Deployment

Our application can be deployed using the scripts we provide:

- `build_docker.sh`: calls `docker build`
- `run_docker.sh`/`run_docker_cluster.sh`: calls `docker run` with the mount flag for accessing the app and resources

It is important, that the dataset folder is placed in the same directory as the `app.py` file. The scripts also have to be called from that directory. The model has to be trained before the application deployment, which can be done by the `model_train.py` script.

Measurements

In our experiment setup, we measure two different times as depicted in figure 1. The `prediction` time spans from the entry to the exit of the `detection_loop()` function. The `request` time is measured at the client and spans from the request to the response. All times were recorded using the `time` module from `Python`.

Since we have decided on uploading the audio files beforehand, this procedure was measured separately. We used `SCP` for uploading the dataset and divided the total upload duration by the amount of files in order to get the average `file upload` time.

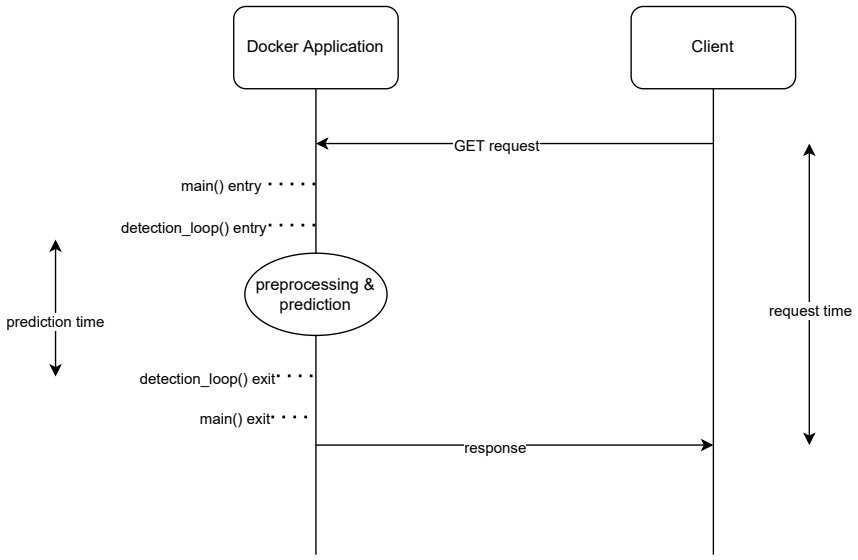

Figure 1: Measurement setup

Experiment setup

The experiment was conducted on a Notebook that is using an Intel Core i7-8565U CPU, while the device was plugged in during the experiment and the power mode in Windows was set to *Best performance*. This device does not use an additional GPU. According to speedtest.net, the network latency is 13 ms, the download bandwidth 35 Mbps and the upload bandwidth 19 Mbps. This test was executed with an active VPN connection to the TU network.

Results

In table 1, we can see the average times as explained before. The inference time is smaller, when using the remote setup. However, this is on average only about 5 ms, whereas the total request duration is 45 ms longer on average. It has to be noted that this does not include the file upload duration, which is about 100 ms per file.

	local	remote
file upload	-	99.8 ms
prediction	62.1 ms	57.4 ms
request	75.8 ms	121.1 ms
total	**75.8 ms**	**220.9 ms**

Table 1: Average times for local & remote setup

In figure 2 and figure 3, we can see boxplots of the different times that we have collected. It can be noted, that the distribution of the time measurements has positive skewness - meaning that there are many outliers with longer times. Additionally, we can characterise the resulting distributions as leptokurtic - meaning that the distribution has most of it's

density at the mode and has “fat tails“.

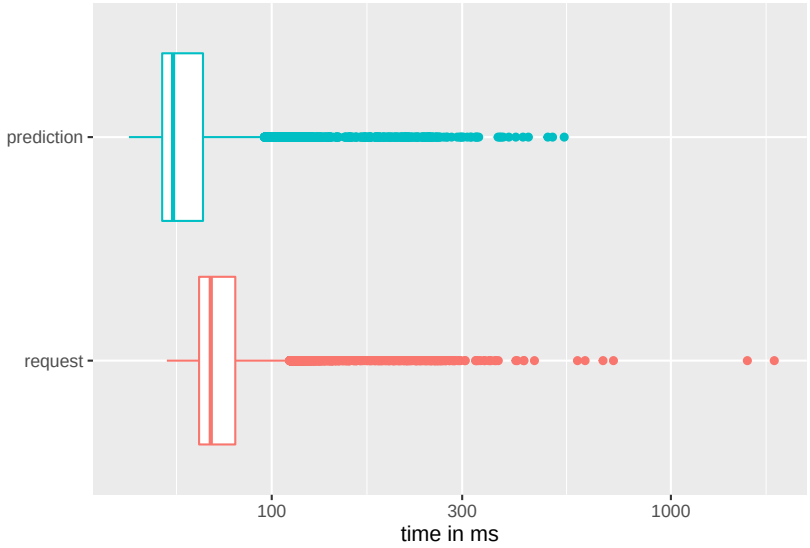

Figure 2: Times with local setup

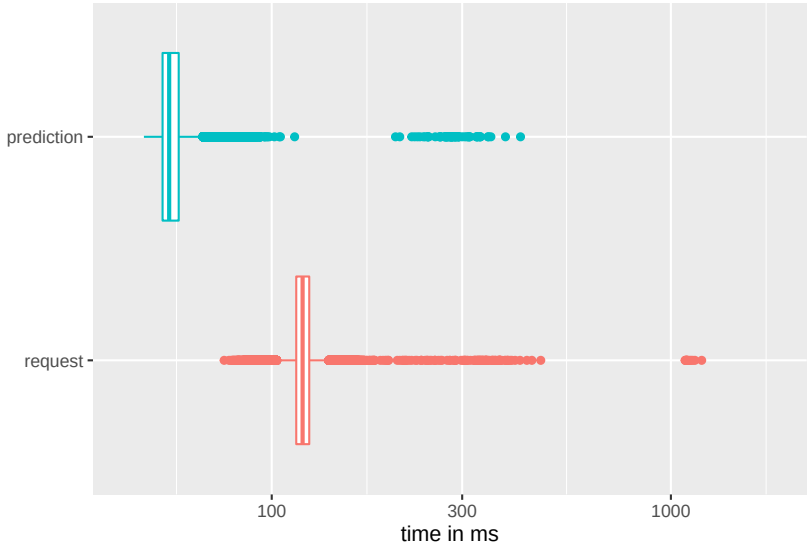

Figure 3: Times with remote setup

Discussion

In our scenario, it is not worth offloading the execution on the remote cluster. Even without considering the file upload time, the request to the remote server is slower than the request to the local application. The file upload alone needs more time than the whole local request. Since the predictions are calculated quickly enough, offloading the execution induces more overhead than improvement. Remote and local execution could be improved

with faster hardware, especially the usage of GPUs. Remote execution can also be further reduced with a smaller physical distance to the server or even connecting directly to the server's network. Tasks that are not too computationally intensive - like the task at hand - can be executed faster locally on modern consumer hardware. However, some cases exist where offloading of the execution could make sense. When using simple and slow hardware, like basic microcontrollers, which in our case would be able to record audio and send network requests, edge computing could have an advantage. Another case we can think of is having computationally more intensive tasks like an extractive QA model for text passages. The added network overhead would then not be as relevant, because of the significant faster inference time.