3.1.1 Local development environment

The local environment has been mainly used for developing the Python scrips and for testing and evaluating trained models. In order to reproduce results, it is recommended to follow the instructions presented in this section. Recreating the exact same development environment is advisable, since it ensures, that everything developed in this thesis will work as intended. For code developement Pycharm 3 (JetBrains, 2017) based on Anaconda (Anaconda Inc., 2017) was used. The operating system is Windows 10. In addition, the qualitative analysis is implemented via Jupyter Notebook (Jupyter Team, 2015). Its unique way of running snippets of code and displaying its output directly beneath the related snippets makes it the ideal tool for qualitative analysis where output has to be investigated manually. All non-standard Python modules can be found in table xy. These modules can be installed via anaconda resp. via pip by the following command:

```
pip install *module-name*
conda install *module-name*
```

Module	Version
tensorflow	4.6.0
Cython	0.13.0
matplotlib	2.1.2
contextlib2	3.2.5
jupyter	1.15.4
pandas	0.22.0
lxml	2.18.4
scikit-learn	0.20.2
pillow	xxxxxx

Table 1. Local Python Modules

In addition to these Python Modules a number of APIs has to be installed in order to use the models developed in this thesis (see table xy).

API	Version
Tensorflow Object Detection API	--
Coco API	--
Google Protobuf	3.4.0

Table 2. Dependencies

Google's object detection API (Huang, et al., 2017) is the core dependency of this thesis. Not only does it provide pre-trained models which are made to be used instantly for a wide range of object detection tasks, it also provides the infrastructure for transfer learning. For a more detailed overview, see section 3.1.3. A direct dependency of the object detection API is the COCO API (Lin, et al., 2014). The COCO API is mainly used for implementing COCO evaluation metrics[1]. According to the Tensorflow object detection API's documentation, the COCO API is optional (Huang, et al., 2019). However, for consistency reasons, I strongly recommend installing it. To install these two APIs, following code has to be executed through git bash[2]:

```
git clone https://github.com/tensorflow/models.git
cd models/research
git clone https://github.com/cocodataset/cocoapi.git
mv cocoapi/PythonAPI/pycocotools .
```

Of course, this step can be as well carried out manually by downloading the repositories and then move the pycocotools folder into the models/research/ directory.

The object detection API partially relies on .proto files, which first have to be compiled into .py files in order to use them. The Protobuf compiler has to be installed by downloading it from its repository[3]. According to (Labinski, 2018) for Windows 10, version 3.4 has to be used. Then, the compiler has to be unzipped. Once it is downloaded and unzipped, execute following command from within the models/research directory to compilate the .proto files:

```
Absolute-path-to-proto-compiler/protoc.exe object_detection/protos/*.proto --
python_out=.
```

As a last step, the libraries have to be added to the environment variable PYTHONPATH (from models/research). This ensures that Python recognizes the corresponding python scripts. This step has to be done each time when running?(plus run setup?)

```
# for bash
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
# for anaconda console
SET PYTHONPATH=%cd%;%cd%\slim
```

[1] For an explanation and discussion of object detection metrics, see section xy.

[2] If not explicitly mentioned, console commands in this section are referring to bash commands. While bash is originally a linux shell, it can also be used on Windows.

[3] https://github.com/protocolbuffers/protobuf/releases/tag/v3.4.0