# 1- Popular frameworks research (week 1)

**Document 1:** **NN most popular libraries.docx** (<https://drive.google.com/open?id=1XH2S-7eN1wl2EYqiN9XQjaOBKwkfuyGq> )

The first step of the study consists of gathering neural network most popular frameworks. After some research on the internet, most results suggested these frameworks:

1) TensorFlow

2) Keras

3) PyTorch

4) Caffe

5) Theano

6) MXNET

7) CNTK

8) DeepLearning4J

9) Caffe2

10) Chainer

11) FastAI

And others

Reference : <https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a>

I referenced other websites in the google document. I also noted each framework’s Github, bug repository and website.

# 2- Mining scripts for issues and comments (week 1 and 2)

**Document 2:** **ml-framework-bugs\script\_issues.py**

From a csv containing frameworks’ information, mines each framework issues (with label or not) and saves them in a json and a csv file.

**Document 3: ml-framework-bugs\script\_comments.py**

From the issues’ csv, mines all comments of all issues in the csv and saves them in a json and a csv file. The labeled issues’ comments will be already in their own csv, since labeled issues are already separated after using script\_issues.py.

# 3- Issues manual reading and Keywords search reading (weeks 2-6 and 6)

*I apologize for the formatting of the tables. Be mindful of the table name, as some “relevant bugs” tables are long.*

**Google Drive folder :** [**https://drive.google.com/open?id=1nSMmAo0kiAlpdIVdbZFeolyICWfQNycH**](https://drive.google.com/open?id=1nSMmAo0kiAlpdIVdbZFeolyICWfQNycH)

**Document 4: Manually reading all frameworks** (<https://drive.google.com/open?id=15KPcTNVlmCPgZum-dQTl-JN7oG2lMZkQ_uss6LdeA7s> )

* TensorFlow (page 1):
* Relevant: manual reading of recent relevant issues. Brief look at the repo to find keywords/aspects some for the classification of bugs.
* Non-relevant: manual reading of recent non-relevant issues. Brief look at the repo to find some keywords/aspects for the classification of bugs.
* Caffe (page 2):
* Relevant:
  + Legitimate issues: manual reading of recent relevant issues. Brief look at the repo to find some keywords/aspects for the classification of bugs.
  + [label:bug] Exhaustive manual reading of Caffe issues with label:bug. The “Our Notes” columns contains Emilio’s opinion of the impact of bugs, which is a better indicator than the grade in the “Issue title” column. From #3254, “Our Notes” columns’ is an estimation of the bug impact. #297 and #284 are attempts to classify issues in the Deep learning stages.
* Non-relevant (page 8): One example of crashing bug. Three possible examples of PR probably having a minor impact on models.
* PR: is not a list of PRs, but a list of keywords.
* Sonnet (page 10):
* Relevant: exhaustive reading of all closed issues. Emilio’s notes in “Our Notes” column.
* Non-relevant: one non-relevant issue that was confusing. No Emilio notes.
* PR: No Emilio notes. I started looking at Sonnet’s PRs, but I did not continue.
* Swift for TensorFlow (page 12):
* Relevant: exhaustive reading of all closed issues. No Emilio notes.
* Non-relevant: non-relevant issues that were confusing. No Emilio notes.
* PR: empty.

**Document 5: manually reading Keras** (<https://drive.google.com/open?id=1ZJHPlkg1C0d9IOj3f6SpoegnSaG1TbGzQBfDgl19umQ> )

* Keras relevant issues (page 1):
* Relevant: manual reading of recent closed relevant issues. Covers a fewer number of issues than PyTorch. I started to focus on Pytorch because of its clearer version’s documentation. No Emilio notes.

**Document 6: Manually reading PyTorch** (<https://drive.google.com/open?id=1m-pJxy1R00Gm4Vvi2lHc6bksKjqL8fmdVxOiaKfG-qo> )

* This document contains commit numbers for almost all issues/commits noted. The version number is also noted for those that was easier to retrace.
* PyTorch relevant issues (page 1):
* Relevant: manual reading of recent closed relevant issues. Covers a good number of issues, but probably not all, because I also used keywords. Most of the issues come from code>releases tabs on GitHub. No Emilio notes.
* PyTorch keywords issues (page 4):
* Relevant: manual reading of numerous issues found using grep and gitlog with keyword “bug” (and possibly “fix” and “bugfix” too…). I read a certain number of issues, then skipped a number of issues to “randomize” the reading. No Emilio notes.
* PyTorch files history, suggestion from the 9th of July meeting (page 10):
* Relevant: exhaustive reading of changes history for conv.py, batchnorm.py, maxpooling.py, pixelshuffle.py and pooling.py. No interesting results for maxpooling and pooling. No Emilio notes, but he said #12952’s previous commit causes a crash.

# 4- Frameworks installation documentation (weeks 4 and 5)

**Google drive folder :** [**https://drive.google.com/open?id=1lOnB1Av7gzzz6vXt\_XXx6iiXVIRLYiZq**](https://drive.google.com/open?id=1lOnB1Av7gzzz6vXt_XXx6iiXVIRLYiZq)

**Document 7:** **releases version support TensorFlow** (<https://docs.google.com/document/d/19T5njgSxdc74wnznbd3BssWKeVLZNSwq2WlBhU26PQc/edit>)

1) TensorFlow versions’ compatibility (gpu) (page 1): dependencies’ version for each TensorFlow gpu version. The table describes which dependencies are needed to install a buggy version.

2) whl packages for each Python version compatible with TensorFlow 1.13 (page 2): whl packages are easier for the installation of the buggy version. I think this table is for TF 1.14, not 1.13 …

3) whl packages for each Python version compatible with TensorFlow 1.14 (page 3): whl packages are easier for the installation of the buggy version.

4) whl packages for each Python version compatible with TensorFlow 1.12 (page 6): whl packages are easier for the installation of the buggy version.

5) whl packages for each Python version compatible with TensorFlow 1.11 (page 8): whl packages are easier for the installation of the buggy version.

6) whl packages for each Python version compatible with TensorFlow 1.10 (page 11): whl packages are easier for the installation of the buggy version.

7) whl packages for each Python version compatible with TensorFlow 1.9 (page 13): whl packages are easier for the installation of the buggy version.

8) whl packages for each Python version compatible with TensorFlow 1.8 (page 15): whl packages are easier for the installation of the buggy version.

The tables for version 1.7 and earlier are not present because the study focuses on bugs corrected from year 2016 and after.

**Document 8: releases version support PyTorch, Caffe and Theano (**<https://docs.google.com/document/d/13JBxRsZd4wkD4ep2BjeJ_Cz8v081srwaAgvoNFsGWvs/edit>**)**

1) PyTorch versions’ compatibility (page 1): dependencies’ version for each PyTorch version. The table describes which dependencies are needed to install a buggy version.

2) whl packages for each Python version compatible with various PyTorch versions (page 2): whl packages are more convenient for the installation of the buggy version. NOTE: The links are for CUDA 7.5

3) Caffe versions’ compatibility (page 4): dependencies’ version for each PyTorch version. The describes which dependencies are needed to install a buggy version.

4) whl packages for each Python version compatible with various Caffe versions (page 4): whl packages are more convenient for the installation of the buggy version.

5) Theano versions’ compatibility (page 5): dependencies’ version for each PyTorch version. The table describes which dependencies are needed to install a buggy version.

6) whl packages for each Python version compatible with various Theano versions (page 5): whl packages are more convenient for the installation of the buggy version.

**Document 9: meeting notes (**[**https://drive.google.com/open?id=1QKWxlSCWA1x8Q611SBamsuXsP\_\_ZtayrkmOCj6AE\_eE**](https://drive.google.com/open?id=1QKWxlSCWA1x8Q611SBamsuXsP__ZtayrkmOCj6AE_eE) **)**

Produced by Emilio. Summarizes all subjects for 17th June 2019, 24th June 2019, 1st July 2019 and 16th July 2019 meetings. A diagram at page 4 describes the workflow used for the research process of the study

# 5- Call frequency of a bugfix (from week 7)

# 5.1- Existing tracers (week 7)

Python seems to have existing tools for tracing. C language doesn’t seem too.

<https://docs.python.org/2/library/trace.html>

<https://docs.python.org/2/library/traceback.html>

<https://pymotw.com/2/trace/>

# 5.2- C++ tracer code (week 7-8)

Files are in **ml-framework-bugs\C Tracer.**

The integration of this code may cause compilation problems related to links edition. Emilio has done work regarding this.

# 5.3- C++ syntax analyzers (week 7-10)

CastXML: <https://github.com/CastXML/CastXML>

<https://github.com/thewtex/CastXMLSuperbuild>

GCC-XML: <https://github.com/gccxml/gccxml>

CastXML Is the maintained version of GCC-XML. Using the Superbuild is much simpler. If you wish to build from source, you will need to install Clang and LLVM … You might want to read these guides to build them:

<http://clang.llvm.org/get_started.html>

<https://www.llvm.org/docs/CMake.html>

# 5.4- Python inserter of the trace call (weeks 8-12)

**Document 10: ml-framework-bugs\2019-07-22 code insterter\python\_insert.py**

The code in python\_auto\_insert shall be integrated in this script. In other words, this inserter should be an automatic inserter.

Command prompt call: python python\_inserter.py (commit\_sha)

Example: python python\_inserter.py efc3d6b65

**Document 11: ml-framework-bugs\2019-07-22 code insterter\python\_auto\_insert.py**

**is\_unindented\_insertable() and analyze\_python\_file() methods requires more testing.** It is not confirmed that they can cover all cases of insertion. The file ml-framework-bugs\2019-07-22 code insterter\test\_dataloader.pypresents many cases of indentation that would be useful in testing.