MIMUW-hats Test plan

Version 0.1, last modified 28.04.2020

Michał Borowski, Michał Makowski, Michał Tyrolski, Szymon Tworkowski

Backend

Using **TypeScript** with a nicely configured linter allows us to catch many errors before passing build - a way of smoke testing. During rapid development phase, mainly manual tests (via Postman-like tools). Some automatic smoke tests to ensure the app is relatively stable between commits - technologies used (**JS** stack):

- mocha
- chai
- sinon

After implementing each of the major functionalities of our API, writing more tests to ensure it works as it should according to specification - more extensive acceptance testing, since these tools allow us to mock database and external APIs. Mostly unit tests performing a bunch of requests on in different order and checking response properties + database state. As mentioned, regression testing will be performed across commits. There is no special need for performance and load testing in our app - the volume of data and amount of users are pretty small even in worst-case assumptions.

Machine Learning

The basis of the machine learning libraries such as **Keras**, **TensorFlow** is math. They are over and over-tested by the authors. The main approaches used there are:
math testing via unit tests of specific mathematical expressions (integrals etc.)
unit tests created on small models without randomized weights to check that the predictions are equal to those from calculations on the sheet.

We assume (reasonable) that the libraries we use work correctly.

The problem of testing the machine learning part is closely related to the concept of model validation on the validation set and on the test set. We separate three parts from the dataset that we have. The first part is the set where the model learns, i.e. modifies his weights to gain the best accuracy. These results are not directly evaluated during the testing process. The second part is the so-called validation set. There is the first part of testing, but also the "boosting" of the model. Seeing the pictures, model boosts some of its parameters. The version that performs best is selected, and each of these versions is evaluated at a given step. At the last of the machine learning core and second stage of testing, we have the test set. The frozen model responds to queries and its answers are compared with real ones. Results from those steps are corresponding to accuracy.

The mechanism described above is repeated every time the model is updated. Each time when some code is modified, then we check if the final evaluation result on the test set is at least **P**. If not, that means our model doesn't perform better than previous ones and we should consider change strategy. Variable P represents minimum accuracy which we are willing to accept. That value can change during the increasing model’s effectivity.

Estimating accuracy algorithm depends on the task. The first one just determines the class of the object on the image. Ist est model should be able to recognize if the given sample represents a hat or not hat. The second task (if we will be able to implement that) is to localize objects on the image. The key action is to make a bounding box of the localized items. This cannot be done clearly so before testing (and whole learning algorithm too) should make his own bounding boxes on training/validation/test set using some software. We will use both prepared bounding boxes on websites and these made on our own. Marking bounding boxes, so-called labeling, can be done using labeling software (open-source: https://github.com/tzutalin/labelImg). If we have data for learning & testing, accuracy is measured using the Jaccard index, also known in the machine learning community as Intersection over Union. That task will probably have a small number of samples so we will use also Cross-Validation algorithms.

Frontend

Testing of the frontend will mainly use the Jest framework built into React, which allows automated testing. Unit testing of components’ functionality can be accomplished by mocking the backend API. The Enzyme library along with Jest’s snapshot testing can be used to test the rendering of components in isolation. Integration testing can also be done by means of these frameworks. End-to-end testing can be done with the Cypress framework in order to test real use-cases and requirements.

Merging

Because of independent of frontend and other parts of project, there is no problem to make tests independently.