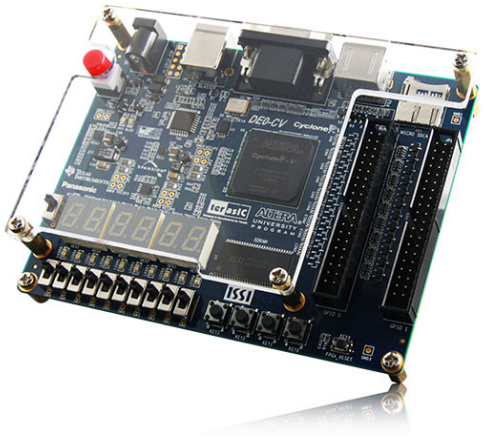
Karlsruher Institut für Technologie

HePICS Deployment-Phase Document

Andres Stober Mehyar Cherni Ibrahim Bouriga
Linjuan Fan Bahaa Mahagne

March 22, 2019



Contents

1 Introduction : HePICS 3

2 Software Engineering as A Practise 4

3 Functionality Description 5

4 Implementation Challenges 6

5 Libraries and Tools 6

6 Requirements Check 7
- 6.1 Must requirements . . . 7
- 6.2 Can requirements . . . 8

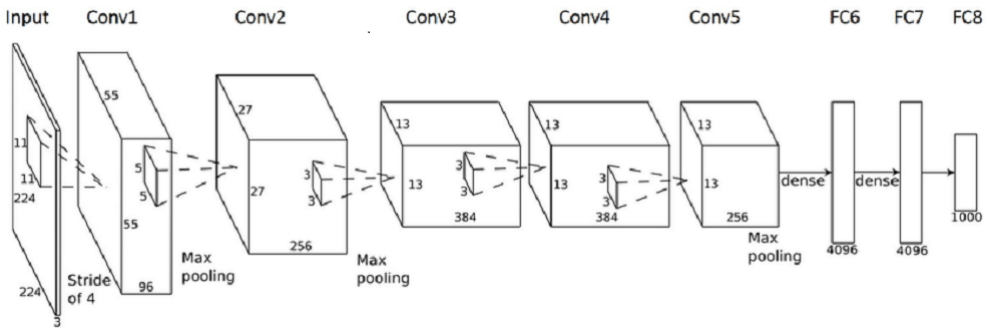
7 Statistics 9

8 Readme and Installation 12

9 Screenshots 13

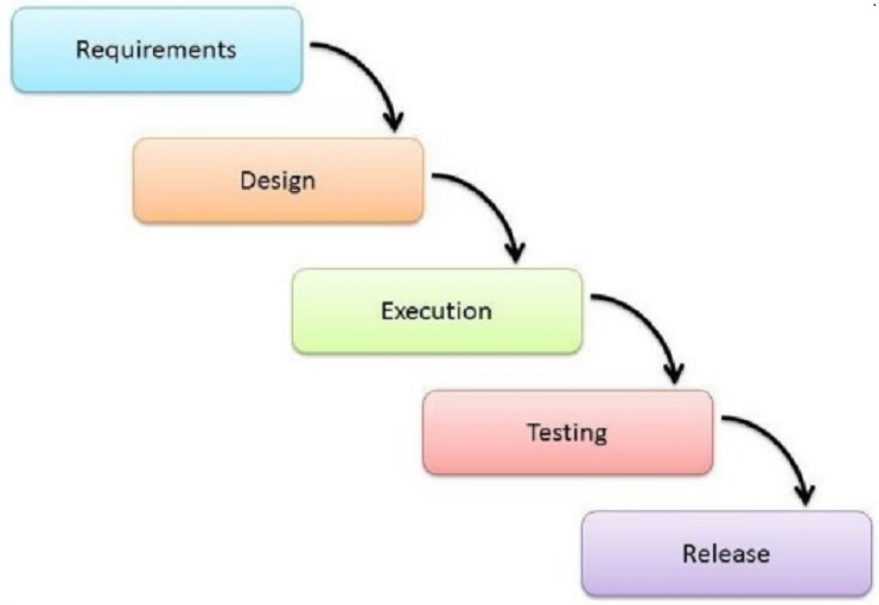
1 Introduction : HePICS

HePICS stands for Heterogeneous Platform for an Image Classification System, which was the task of this software engineering practise.
The software employs the topology of the AlexNet artificial neural network in order to classify images. For this purpose, it offers the user the possibility of choosing one or multiple images, enabling the platforms it is designed to run on, as well as the operation mode. Besides, it contains the feature of aggregating the results of different classifications. A user inteface has been designed to make the contact with the software easier and its use more appealing. Every step the user has to take to properly run the classification is lead to by a button .
The software only enables the classification through a cpu for the moment, which means the choice of the mode doesn't have any influence on the performance. The classification of a single image lasts in average around 4,6 seconds. Although running on the fpga should theoretically increase the performance.
The progressbar gives an estimation about the duration of the classification of all input images, and after testing different use cases, the software has been proven to be in a stable state and to run as expected.



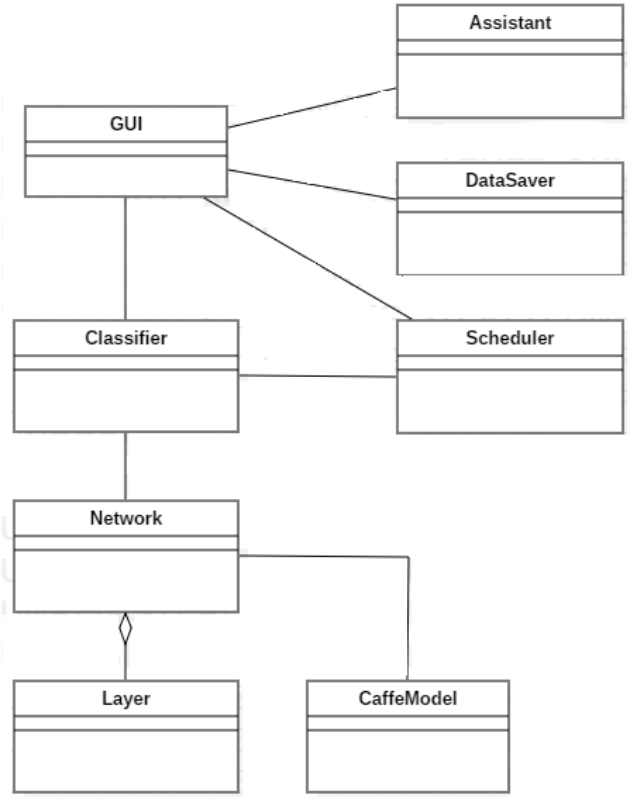
2 Software Engineering as A Practise

Software engineering is a practise which sets the rules for each individual or group who is willing to develop a software product. During the development of HePICS, the waterfall model has been followed, which leads us to this final phase : the release, or deployment. It has been made sure that the common knowledge of software engineering was applied during every phase, was it analyzing the requirements, designing and implementing the structure or testing it, or even version control. To read about these practises seems like any other knowledge, but to perform them has proven to be a harder task than expected, and definitely an interesting experience. As a matter of fact, besides object oriented programming, we had to explore every corner of the waterfall model, from UML diagrams to unit tests and deployment, the idea around software engineering has been made clearer.



3 Functionality Description

The following only describes the general structure of the program, and the major changes made after the design phase.
The program implements with the help of the caffe library an AlexNet neural network. The network is generated automatically through the configuration file when the program is run. It contains an array of layers set in the right order of the topology. Possible layer types are convolutional, normal response normalization, maxpool, or fully connected. The activation functions have also been implemented as layers in order to make the design of a pipeline possible.
The GUI takes the input from the user and passes it through the Assistant, Scheduler and DataSaver to the system. The Classifier takes then each input and passes it through the layers' pipeline, which produces a result to display on the user interface. If the fpga is enabled, the classifier should run the convolutional layer on it. Unfortunately, its implementation couldn't be accomplished in time.



4 Implementation Challenges

Due to the group being unfamiliar with the concept of a neural network, loading images correctly has proven to be quite the challenge. The first interpretation was a simple 2D vector, but we slowly figured out it wasn't enough since the convlutional layer needs more paremeters for it to apply its filter. While refactoring the program and trying to code a cleaner design after the implementation phase, the image was represented as a 3D model.
The program has been able to run since then, but with wrong results. While debugging, errors in the implementation of some layers were uncovered, but the major issue has been figured with the help of the caffe model. In fact reading its implementation and following each step of it has been nothing but helpful to the process of correctly interpreting the classification.
Another task that turned out to be a bit challenging was the the use of external libraries on eclipse, as it demands more steps than library integration with programming languages we're familiar with.

5 Libraries and Tools

Eclipse was used to manage the project and building it, however QtCreator was also used to design the user interface. Both were later integrated on eclipse.
These are the libraries used in the program :

- Qt5Gui : contains the QImage class which was used to read and load images.
- Qt5Core : contains the QString class which is necessary to specify the input path
- Qt5Widgets : contains the classes necessary for creating a graphical user interface
- Protobuf : used to serialize strcutured data
- Caffe : served as a parser for the neural network and generated the weight file.
- GoogleTest : provided unit testing functions

6 Requirements Check

6.1 Must requirements

The following must requirements have been fulfilled.

- The system must be able to classify images.
- The system must allow the selection of one or more input images.
- The system must show the result of the treatment.
- The system must employ artificial neural networks.
- The system must deploy the AlexNet deep neural network.
- The system must know at least one set of weights.
- The system must offer three performance profiles :
  - High Performance
  - Low Power
  - Energy Efficency
- The system must allow the control and the execution of commands through a GUI interface.
- The system must be able to run the GUI on an Ubuntu system.
- The system must exhibit a classification interface.
- The system must include the aggregate feature. This feature fuses the results of different treatments of the same object in order to reach more precision.
- The system must offer the possibility of entering up to 5 images in the aggregate feature.

The following must requirements have not been fulfilled.

- The system must be able to execute intense calculations through heterogeneous platforms (CPU, GPU, FPGA, ASIC).
- The system must support an FPGA through OpenCL.
- The system must hide the OpenCL details behind an abstraction layer.
- The system must be able to measure its performance.
- The system must be able to measure its power consumption.
- The system communicate with another system. It looks into a file to search for an image treatment request, and send back the results via Ethernet connection.

6.2 Can requirements

The following can requirements have been fulfilled.

- The system can offer the possibility of entering more than 5 images in the aggregate feature.
- The system can run batch processing.
- The system can pause, resume or cancel the classification process.
- The system can beautify the output result.

7 Statistics

The following presents some numbers and statistics about the project's implementation and testing.

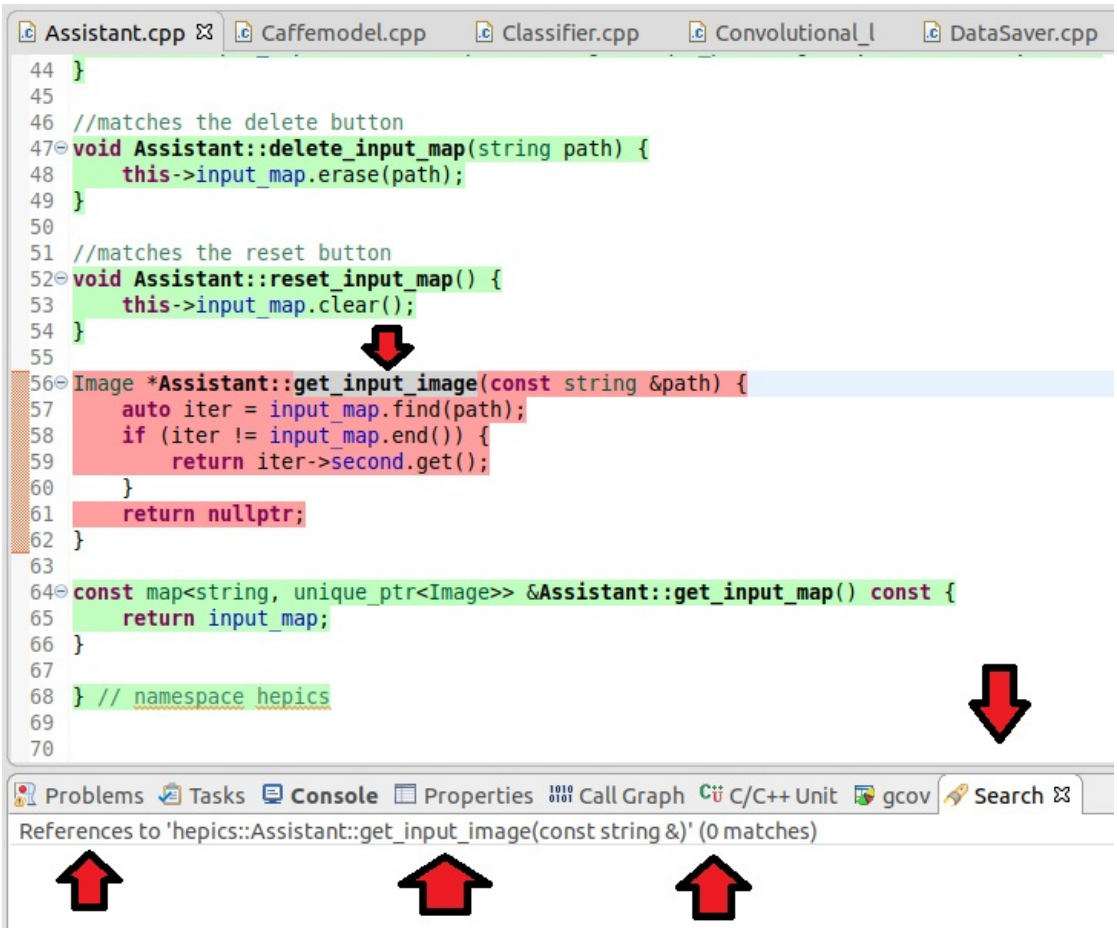
- Lines of code : around 3400 in total, around 500 lines of comments or whitespaces
- Branches : Master which contains the presentations and documents + old code, another 13 branches which served to the integration, or individual work
- Commits : 340
- Tests : 15 Test classes were written, leading to more than 30 test cases
- Coverage : reached an overall of 68 %

Most parts which weren't tested are either very simple implementations, dead code, or exception handling. Some of these have been manually tested, however, every layer and functions which are to heavily to rely on have been properly tested.

Name	Total Lines	Instrumented	Executed Line	Coverage %
hepics	1,410	673	459	68.2%
Assistant.cpp	69	34	29	85.29%
Caffemodel.cpp	176	88	76	86.36%
Caffemodel.h	45	5	1	20.0%
Classifier.cpp	118	72	1	1.39%
Convolutional_layer.cpp	84	50	46	92.0%
Convolutional_layer.h	29	2	1	50.0%
DataSaver.cpp	140	81	55	67.9%
DataSaver.h	44	1	0	0.0%
Exception.cpp	7	2	0	0.0%
Exception.h	8	1	1	100.0%
Fully_connected_layer.cpp	42	25	20	80.0%
Fully_connected_layer.h	20	2	1	50.0%
Function_layer.cpp	107	64	62	96.88%
Function_layer.h	17	2	2	100.0%
Image.cpp	52	29	26	89.66%
Image.h	61	9	8	88.89%
Layer.h	19	2	2	100.0%
Local_response_normalization_layer.cp	51	27	27	100.0%
Local_response_normalization_layer.h	8	1	1	100.0%
Maxpool_layer.cpp	52	29	28	96.55%
Maxpool_layer.h	10	1	1	100.0%
Network.cpp	16	6	4	66.67%
Result.cpp	40	25	21	84.0%
Scheduler.cpp	195	115	46	40.0%

Caffemodel.cpp | Classifier.cpp | Convolutional_l | Fully_connected | Netw

```
125 static void read_message_from_file(Message &message, const string &path) {
126     auto file = make_unique<File>(path, O_RDONLY);
127     auto fis = make_unique<FileInputStream>(file->fd);
128     auto cis = make_unique<CodedInputStream>(fis.get());
129     cis->SetTotalBytesLimit(INT_MAX, INT_MAX / 4);
130     if (!message.ParseFromCodedStream(cis.get())) {
131         throw Parse_failed { };
132     }
133 }
134
135 vector<unique_ptr<Layer>> Model::parse_layers(const string &path) {
136     auto layers = vector<unique_ptr<Layer>>();
137     auto net_parameter = NetParameter { };
138     read_message_from_file(net_parameter, path);
139     for (auto &param : net_parameter.layer()) {
140         auto hepics_layer = make_hepics_layer(param);
141         if (hepics_layer != nullptr) {
142             layers.push_back(move(hepics_layer));
143         }
144     }
145     return layers;
146 }
147
148 unique_ptr<Image> Mean::parse_mean(const string &path) {
149     auto blob = BlobProto { };
150     read_message_from_file(blob, path);
151     return make_image_from_blob(blob);
152 }
153
154 const char *Model_exception::what() const noexcept {
155     return "hepics::Model_exception";
156 }
157
158 const char *Open_failed::what() const noexcept {
159     return "hepics::Open_failed";
160 }
```

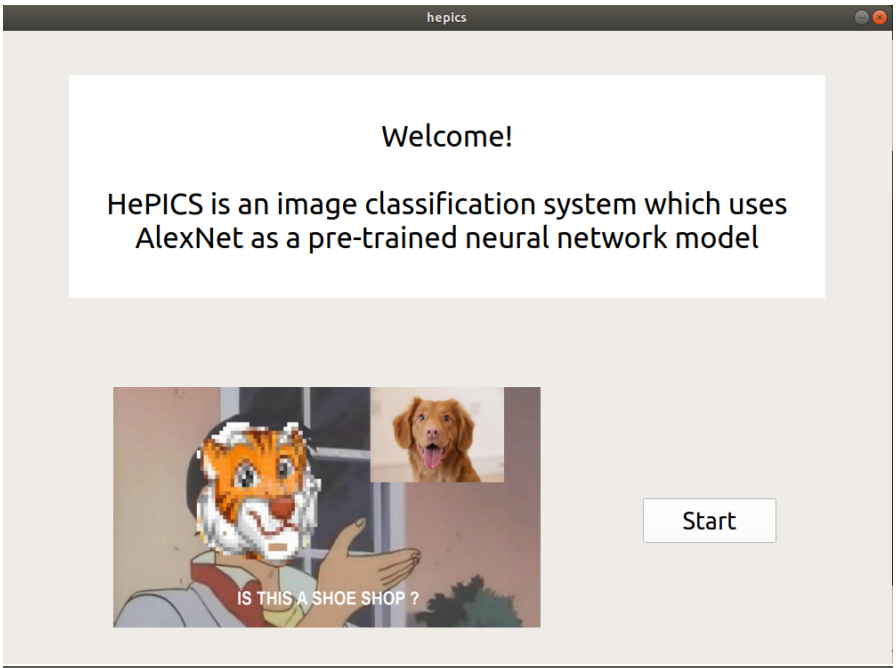

Assistant.cpp
Caffemodel.cpp
Classifier.cpp
Convolutional_l
DataSaver.cpp
}
//matches the delete button
void Assistant::delete_input_map(string path) {
this->input_map.erase(path);
}
//matches the reset button
void Assistant::reset_input_map() {
this->input_map.clear();
}
Image *Assistant::get_input_image(const string &path) {
auto iter = input_map.find(path);
if (iter != input_map.end()) {
return iter->second.get();
}
return nullptr;
}
const map<string, unique_ptr<Image>> &Assistant::get_input_map() const {
return input_map;
}
} // namespace hepics
Problems
Tasks
Console
Properties
Call Graph
C/C++ Unit
gcov
Search
References to 'hepics::Assistant::get_input_image(const string &)' (0 matches)

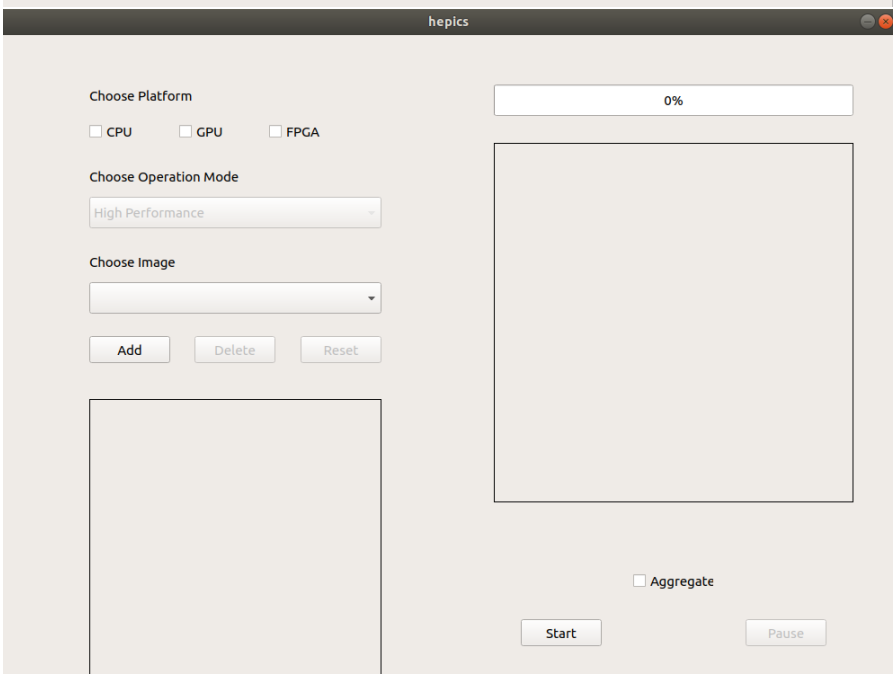
8 Readme and Installation

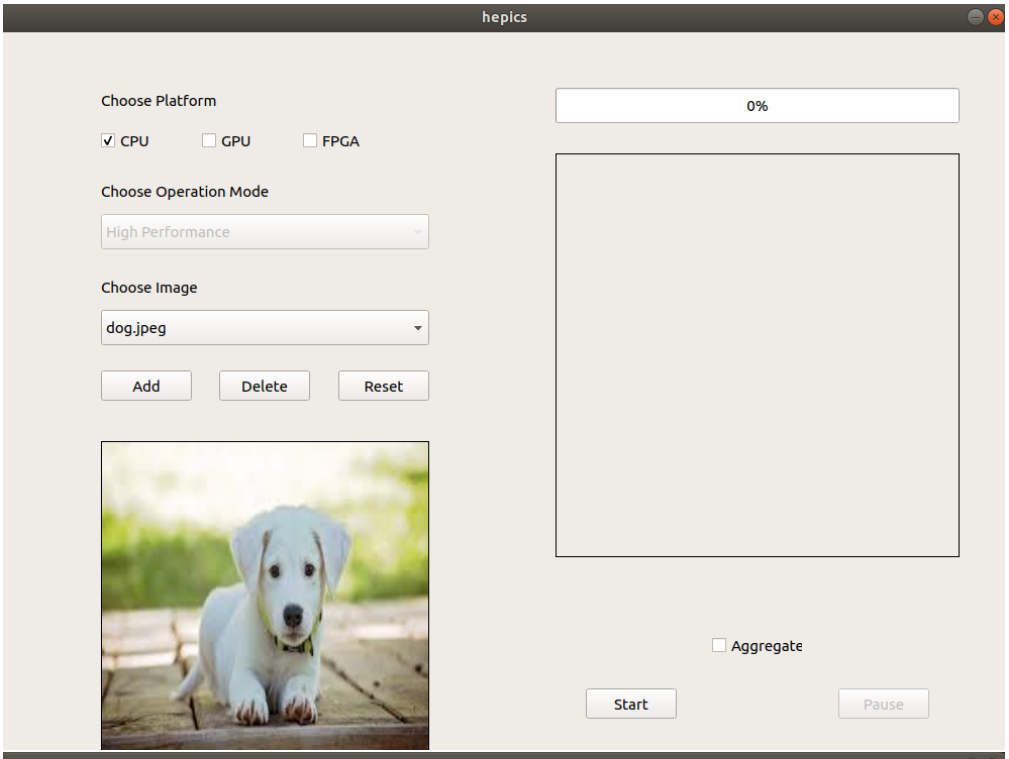
To download the package, please run the following command in the terminal :
scp pselab1819@141.3.77.54: /hepics.tar.gz hepics.tar.gz
password : pselab1819
Once the package has been unzipped, please refer to the instructions in the readme file.
Installation :

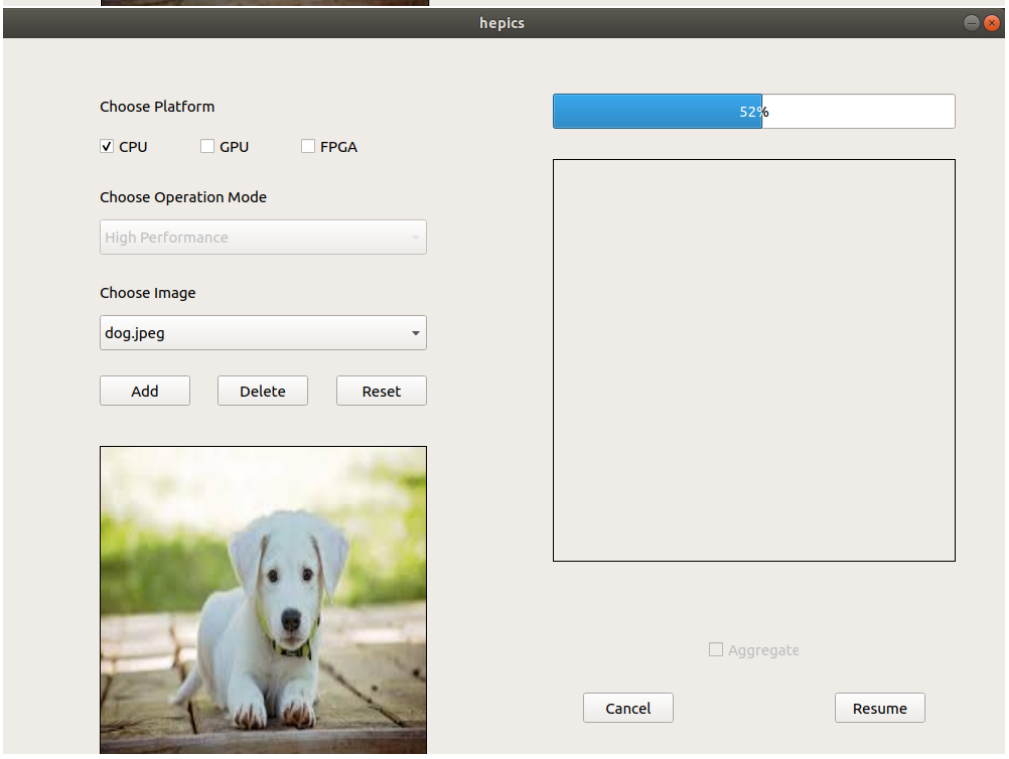
- sudo apt-get install libprotobuf-dev protobuf-compiler
- sudo apt-get install build-essential
- sudo apt-get install qtcreator
- sudo apt-get install qt5-default
- make all
- hepics

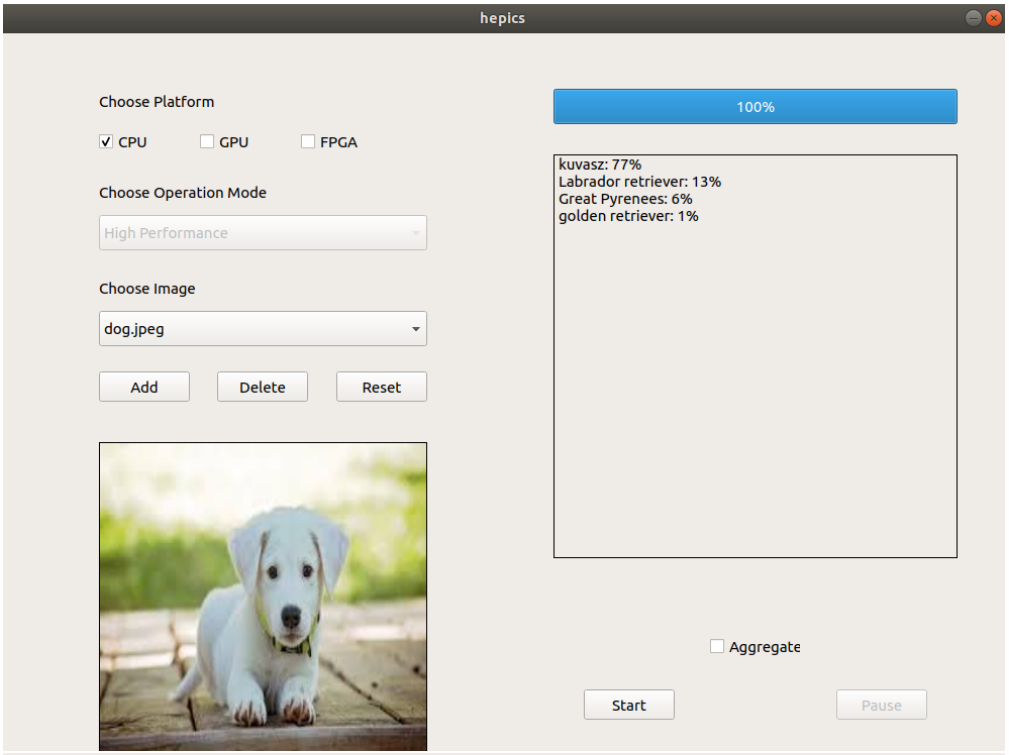
9 Screenshots

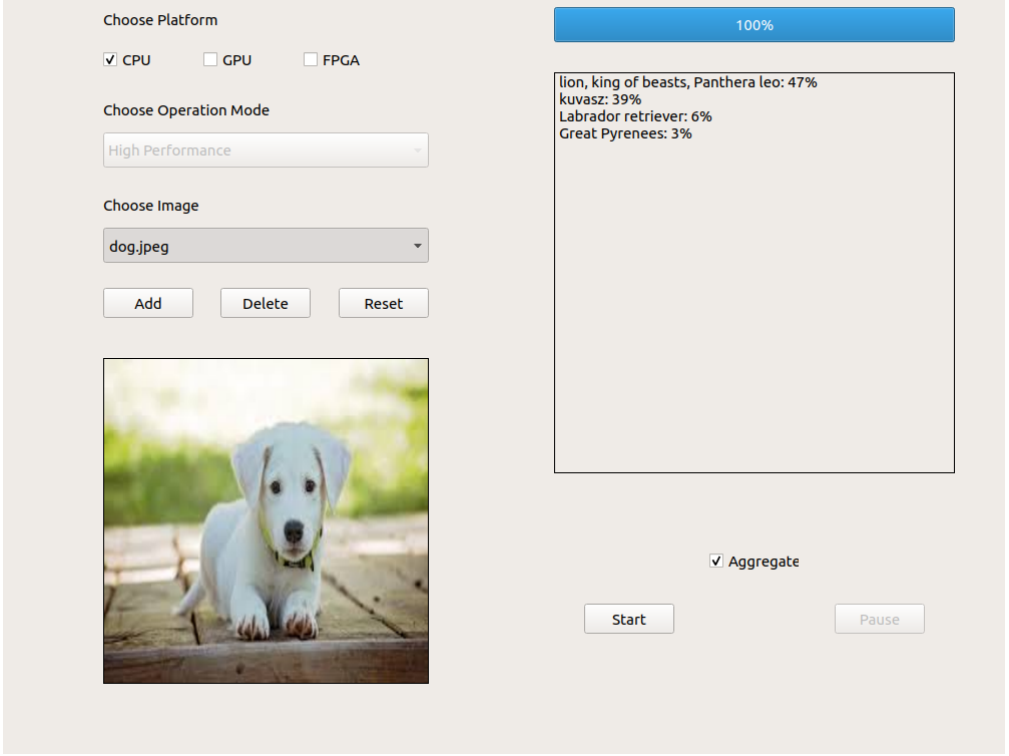





hepics
Choose Platform
CPU
GPU
FPGA
Choose Operation Mode
High Performance
Choose Image
dog.jpeg
Add
Delete
Reset
0%
Aggregate
Start
Pause


hepics
Choose Platform
CPU
GPU
FPGA
Choose Operation Mode
High Performance
Choose Image
dog.jpeg
Add
Delete
Reset
52%
Aggregate
Cancel
Resume


hepics
Choose Platform
CPU
GPU
FPGA
Choose Operation Mode
High Performance
Choose Image
dog.jpeg
Add
Delete
Reset
100%
kuvasz: 77%
Labrador retriever: 13%
Great Pyrenees: 6%
golden retriever: 1%
Aggregate
Start
Pause


Choose Platform
CPU
GPU
FPGA
Choose Operation Mode
High Performance
Choose Image
dog.jpeg
Add
Delete
Reset
100%
lion, king of beasts, Panthera leo: 47%
kuvasz: 39%
Labrador retriever: 6%
Great Pyrenees: 3%
Aggregate
Start
Pause