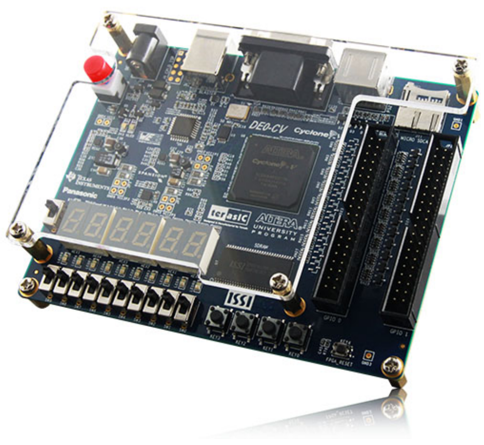
HePICS Test-Phase Document

Andres Stober Mehyar Cherni Ibrahim Bouriga
Linjuan Fan Bahaa Mahjane

March 1, 2019



Contents

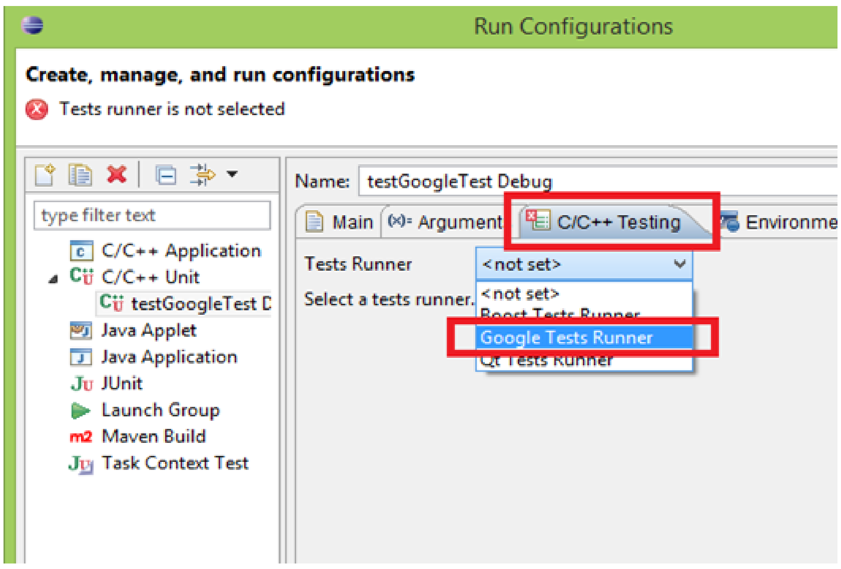
1 Introduction

For the purpose of testing submodules and base classes of our program, we configured GoogleTest on eclipse. As more tests were run, it turned out that the integration of the submodules was of a bigger difficulty than estimated, which is why we decided to re-structure the program, in a way we can use the old code, and make the integration of different parts an easier and safer process at the same time.
We oriented ourselves as much as possible to the design set during the design phase, and of course, with minor changes, our goal has been mostly achieved, since unit tests are all successful.
As much as this is seen as an achievment, it is safe to say and keep in mind that tests are run to uncover errors, not to prove their absence. Therefore, the effectiveness of these tests will be proven in the next phase, when the full integration is finished.

2 Google Test

Google Test is a unit testing library for the C++ programming language, based on the xUnit architecture. This section describes reasons why we decided to use this framework.

- Google Test is designed to be portable and it works around various bugs in various compilers and environments
- Google's test framework has built-in assertions that are deployable in software where exception handling is disabled
- Running the tests is simple and it's easy to write assertions that generate informative messages
- Google Test automatically detects your tests and doesn't require you to enumerate them in order to run them.
- Google's test framework provides excellent support for handling such situations. You can repeat the same test as many times as neededs using the Google framework. See 1


Run Configurations

Create, manage, and run configurations

Tests runner is not selected

type filter text

C/C++ Application
C/C++ Unit
testGoogleTest D
Java Applet
Java Application
JUnit
Launch Group
Maven Build
Task Context Test

Name: testGoogleTest Debug

Main | Argument | C/C++ Testing | Environme

Tests Runner <not set>

Select a tests runner.

<not set>
Boost Tests Runner
Google Tests Runner
Qt Tests Runner

Figure 1: google test

3 Tests

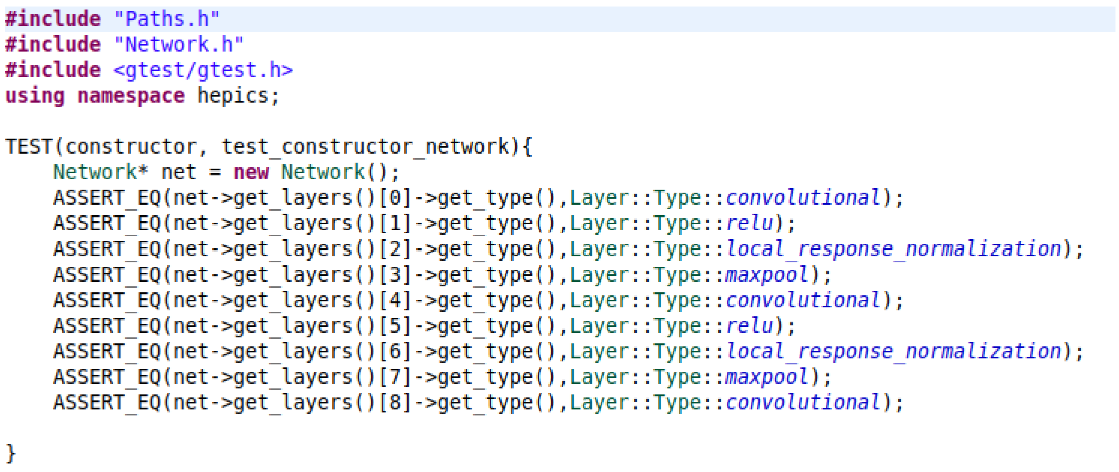
The following displays the tests that were run, and their results.

3.1 Network

The network consists of a vector of multiple of layers, many of which were tested. The activation layers were not tested as they are simple mathematic functions. Example : relu, sigmoid

3.1.1 Network

The constructor of Network loads the layers from a configuration file through Caffe parser. Therefore, tests were run to check if these layers are properly loaded in the class. See figure 2



```
#include "Paths.h"
#include "Network.h"
#include <gtest/gtest.h>
using namespace hepics;

TEST(constructor, test_constructor_network){
    Network* net = new Network();
    ASSERT_EQ(net->get_layers()[0]->get_type(),Layer::Type::convolutional);
    ASSERT_EQ(net->get_layers()[1]->get_type(),Layer::Type::relu);
    ASSERT_EQ(net->get_layers()[2]->get_type(),Layer::Type::local_response_normalization);
    ASSERT_EQ(net->get_layers()[3]->get_type(),Layer::Type::maxpool);
    ASSERT_EQ(net->get_layers()[4]->get_type(),Layer::Type::convolutional);
    ASSERT_EQ(net->get_layers()[5]->get_type(),Layer::Type::relu);
    ASSERT_EQ(net->get_layers()[6]->get_type(),Layer::Type::local_response_normalization);
    ASSERT_EQ(net->get_layers()[7]->get_type(),Layer::Type::maxpool);
    ASSERT_EQ(net->get_layers()[8]->get_type(),Layer::Type::convolutional);

}
```

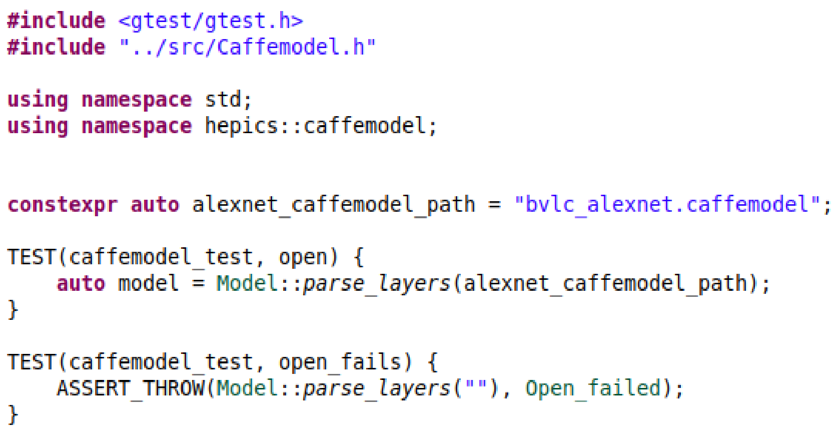
Figure 2: network test

3.1.2 Caffe

Caffe was used to access functions which made the network model easier to implement, since it offers parsing methods to fill each layer with its proper fields. It also provided as the weights. See **??**

3.1.3 Convolutional Layer

The convolutional layer has one of the most complicated implementations of the network model. 4 test cases were run to make sure it is running the way it should.



```
#include <gtest/gtest.h>
#include "../src/Caffemodel.h"

using namespace std;
using namespace hepics::caffemodel;


constexpr auto alexnet_caffemodel_path = "bvlc_alexnet.caffemodel";

TEST(caffemodel_test, open) {
    auto model = Model::parse_layers(alexnet_caffemodel_path);
}

TEST(caffemodel_test, open_fails) {
    ASSERT_THROW(Model::parse_layers(""), Open_failed);
}
```

Figure 3: caffe test

Convolution The idea of the convolution is to go through all image pixels and use the filter on them. However the filter might not cover the right pixels that's why we need to check in every iteration if the latter is within the image. Other important things to be considered in this procedure. For instance the caffe model of the layer itself, which provide further specification other than usual channels, height and width.

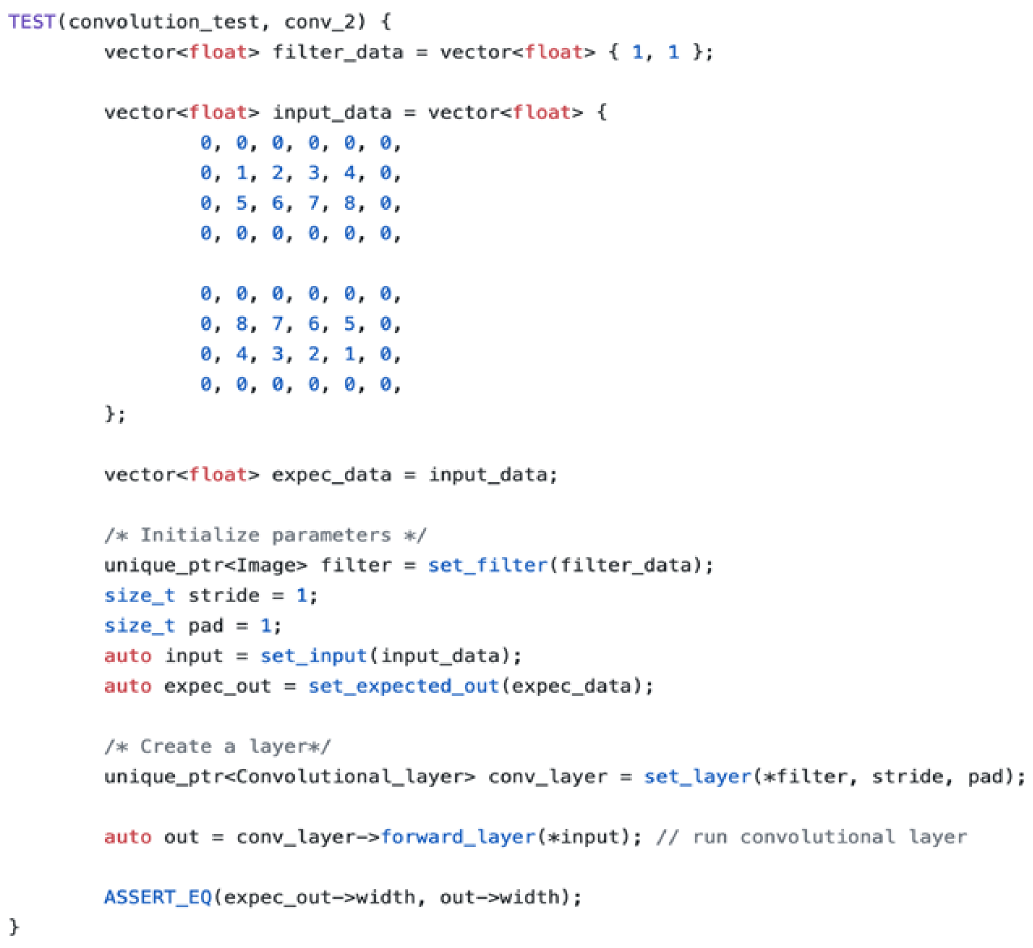
ReLU The caffe model suggest also that the activation function should be considered as a layer too. In order to achieve this approach, we just pass the output of the convolutional as an input for this layer. The function will be applied on the image data and the dimensions of the output do not changes.

Test In order to avoid any unexpected behaviour of the convolutional layer we apply different filters with diffrerent strides on the input. Differents paddings were also used by the input. However we model the image as a set of data and do not use real images in this approach, so that we can avoid interpreting graphical representation by the output. The data are represented as vectors of float. The expected output are calculated manually and compared to the one produced by the layer. See 4

3.1.4 Maxpool Layer

Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. See 5

For each of these tests we created an input image, an expected output and an object of max pool layer. After adding the data for each object we tested with the assertion method



```
TEST(convolution_test, conv_2) {
        vector<float> filter_data = vector<float> { 1, 1 };

        vector<float> input_data = vector<float> {
                0, 0, 0, 0, 0, 0,
                0, 1, 2, 3, 4, 0,
                0, 5, 6, 7, 8, 0,
                0, 0, 0, 0, 0, 0,

                0, 0, 0, 0, 0, 0,
                0, 8, 7, 6, 5, 0,
                0, 4, 3, 2, 1, 0,
                0, 0, 0, 0, 0, 0,
        };

        vector<float> expec_data = input_data;

        /* Initialize parameters */
        unique_ptr<Image> filter = set_filter(filter_data);
        size_t stride = 1;
        size_t pad = 1;
        auto input = set_input(input_data);
        auto expec_out = set_expected_out(expec_data);

        /* Create a layer*/
        unique_ptr<Convolutional_layer> conv_layer = set_layer(*filter, stride, pad);

        auto out = conv_layer->forward_layer(*input); // run convolutional layer

        ASSERT_EQ(expec_out->width, out->width);
}
```

Figure 4: convolution test

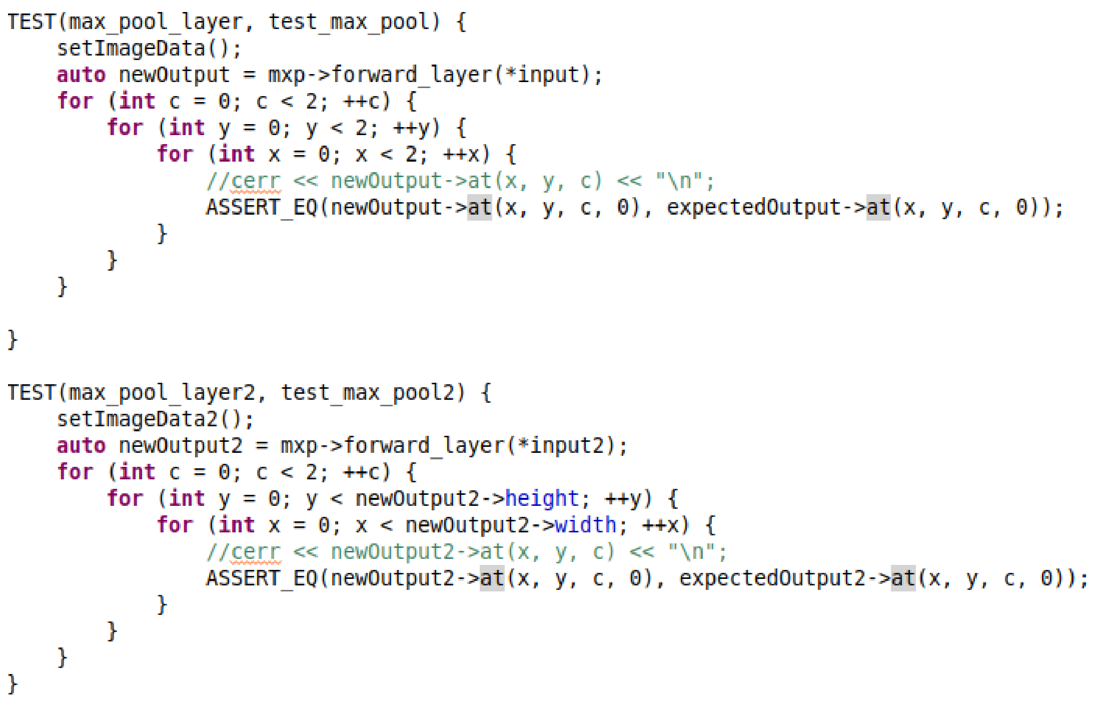
if the output of the max pool layer as we expect it in the output that we created.

3.1.5 Local Response Normalization

See 6

3.1.6 Fully Connected

This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number

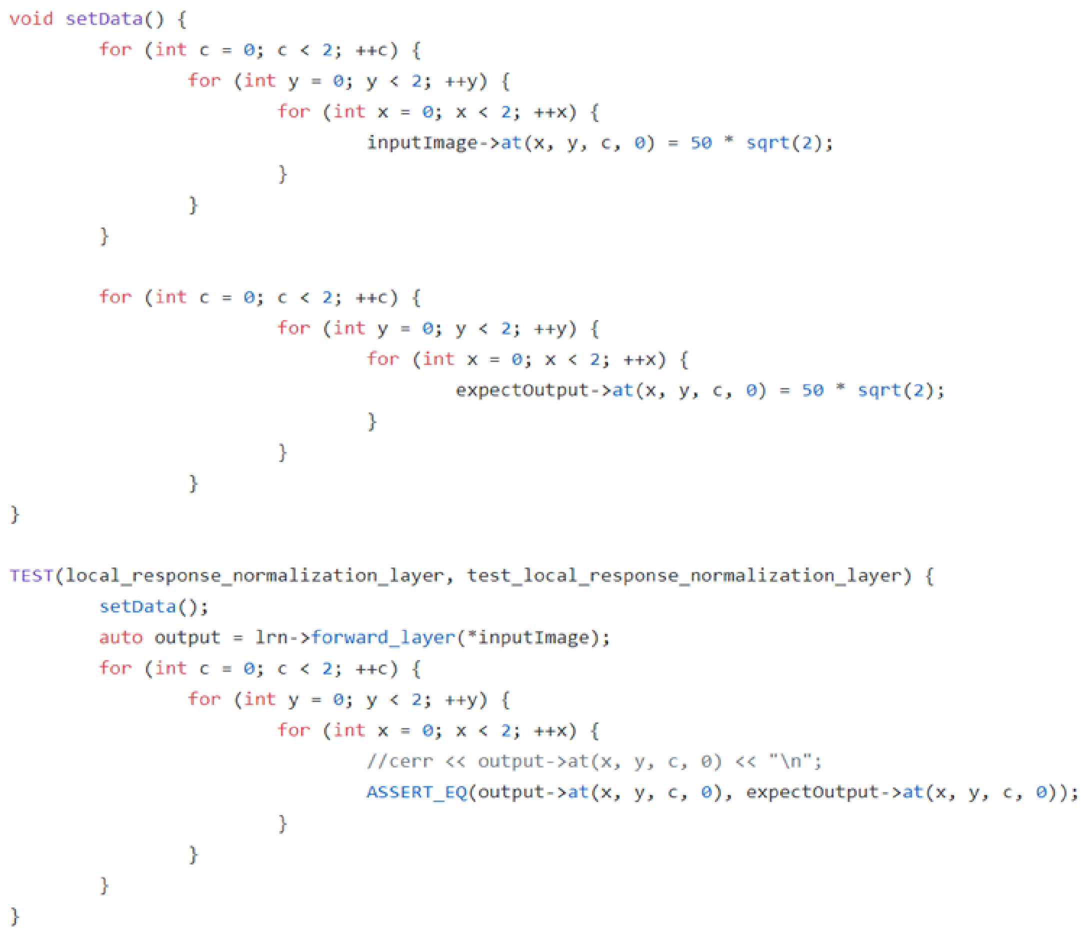


```
TEST(max_pool_layer, test_max_pool) {
    setImageData();
    auto newOutput = mxp->forward_layer(*input);
    for (int c = 0; c < 2; ++c) {
        for (int y = 0; y < 2; ++y) {
            for (int x = 0; x < 2; ++x) {
                //cerr << newOutput->at(x, y, c) << "\n";
                ASSERT_EQ(newOutput->at(x, y, c, 0), expectedOutput->at(x, y, c, 0));
            }
        }
    }

}

TEST(max_pool_layer2, test_max_pool2) {
    setImageData2();
    auto newOutput2 = mxp->forward_layer(*input2);
    for (int c = 0; c < 2; ++c) {
        for (int y = 0; y < newOutput2->height; ++y) {
            for (int x = 0; x < newOutput2->width; ++x) {
                //cerr << newOutput2->at(x, y, c) << "\n";
                ASSERT_EQ(newOutput2->at(x, y, c, 0), expectedOutput2->at(x, y, c, 0));
            }
        }
    }
}
```

Figure 5: maxpool test

of classes that the program has to choose from.

After creating this objets and adding the data to each object we tested if the expected output is equals to the output of the connected layer with the assertion method. See 7



```
void setData() {
    for (int c = 0; c < 2; ++c) {
        for (int y = 0; y < 2; ++y) {
            for (int x = 0; x < 2; ++x) {
                inputImage->at(x, y, c, 0) = 50 * sqrt(2);
            }
        }
    }

    for (int c = 0; c < 2; ++c) {
            for (int y = 0; y < 2; ++y) {
                for (int x = 0; x < 2; ++x) {
                    expectOutput->at(x, y, c, 0) = 50 * sqrt(2);
                }
            }
        }
}

TEST(local_response_normalization_layer, test_local_response_normalization_layer) {
    setData();
    auto output = lrn->forward_layer(*inputImage);
    for (int c = 0; c < 2; ++c) {
        for (int y = 0; y < 2; ++y) {
            for (int x = 0; x < 2; ++x) {
                //cerr << output->at(x, y, c, 0) << "\n";
                ASSERT_EQ(output->at(x, y, c, 0), expectOutput->at(x, y, c, 0));
            }
        }
    }
}
```

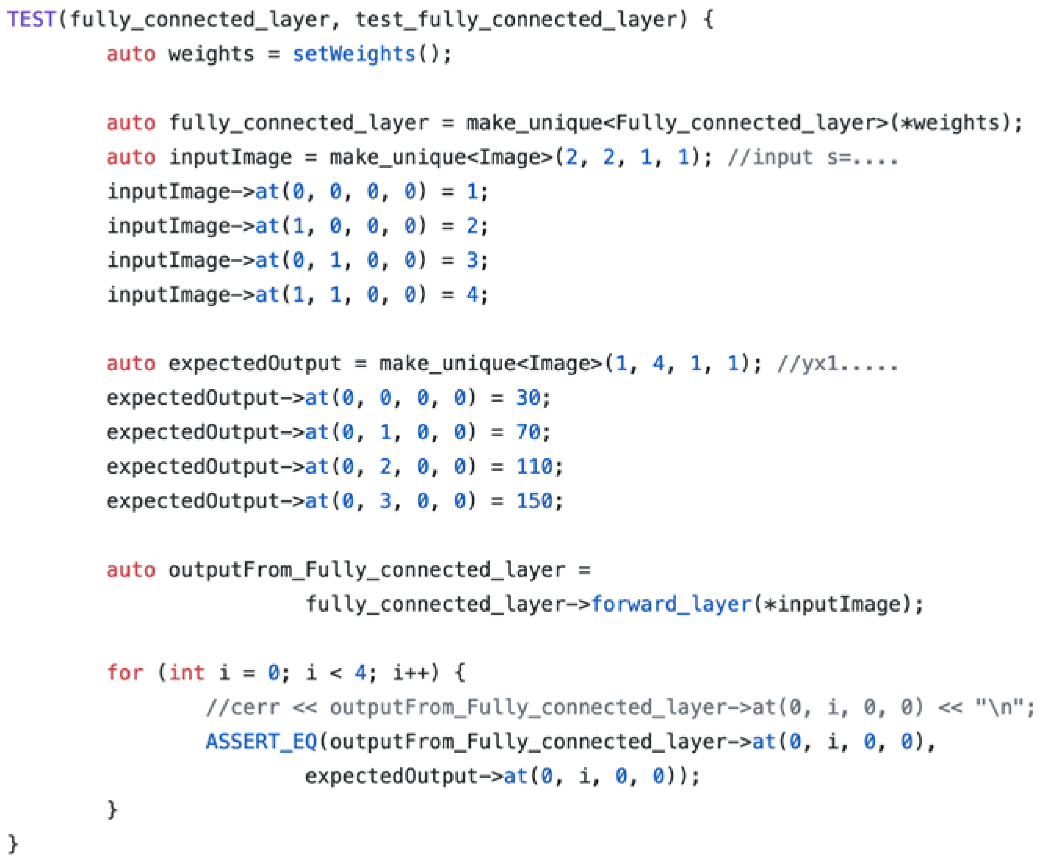
Figure 6: local response test

3.2 Input Output Management

These classes work directly with the user interface, and provide through it the input images and platforms, as well as the mode to the back end , and offer access to the results, whether seperately, or aggregated.

3.2.1 Assistant

The assistant has the input images stored in a map, with their respective paths. See 8



```
TEST(fully_connected_layer, test_fully_connected_layer) {
        auto weights = setWeights();

        auto fully_connected_layer = make_unique<Fully_connected_layer>(*weights);
        auto inputImage = make_unique<Image>(2, 2, 1, 1); //input s=....
        inputImage->at(0, 0, 0, 0) = 1;
        inputImage->at(1, 0, 0, 0) = 2;
        inputImage->at(0, 1, 0, 0) = 3;
        inputImage->at(1, 1, 0, 0) = 4;

        auto expectedOutput = make_unique<Image>(1, 4, 1, 1); //yx1.....
        expectedOutput->at(0, 0, 0, 0) = 30;
        expectedOutput->at(0, 1, 0, 0) = 70;
        expectedOutput->at(0, 2, 0, 0) = 110;
        expectedOutput->at(0, 3, 0, 0) = 150;

        auto outputFrom_Fully_connected_layer =
                        fully_connected_layer->forward_layer(*inputImage);

        for (int i = 0; i < 4; i++) {
                //cerr << outputFrom_Fully_connected_layer->at(0, i, 0, 0) << "\n";
                ASSERT_EQ(outputFrom_Fully_connected_layer->at(0, i, 0, 0),
                        expectedOutput->at(0, i, 0, 0));
        }
}
```

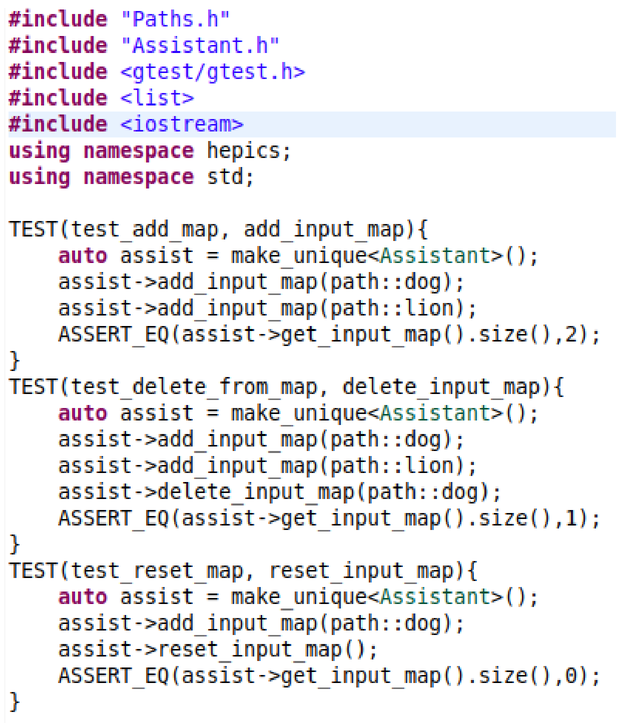
Figure 7: fully connected test

3.2.2 Result

This is a basic representation of the results stored in a vector of pairs, sorted descendently.

3.2.3 DataSaver

The datasaver saves outputs of the classification, converts them into result objects and saves them into a map, directly providing access to them. It also can aggregate the current stored results. See 9

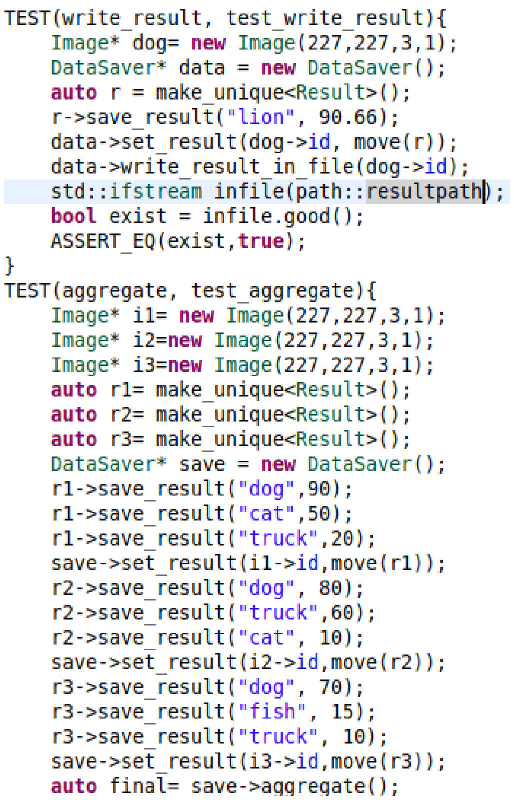


```
#include "Paths.h"
#include "Assistant.h"
#include <gtest/gtest.h>
#include <list>
#include <iostream>
using namespace hepics;
using namespace std;

TEST(test_add_map, add_input_map){
    auto assist = make_unique<Assistant>();
    assist->add_input_map(path::dog);
    assist->add_input_map(path::lion);
    ASSERT_EQ(assist->get_input_map().size(),2);
}
TEST(test_delete_from_map, delete_input_map){
    auto assist = make_unique<Assistant>();
    assist->add_input_map(path::dog);
    assist->add_input_map(path::lion);
    assist->delete_input_map(path::dog);
    ASSERT_EQ(assist->get_input_map().size(),1);
}
TEST(test_reset_map, reset_input_map){
    auto assist = make_unique<Assistant>();
    assist->add_input_map(path::dog);
    assist->reset_input_map();
    ASSERT_EQ(assist->get_input_map().size(),0);
}
```

Figure 8: assistant test

3.2.4 Scheduler

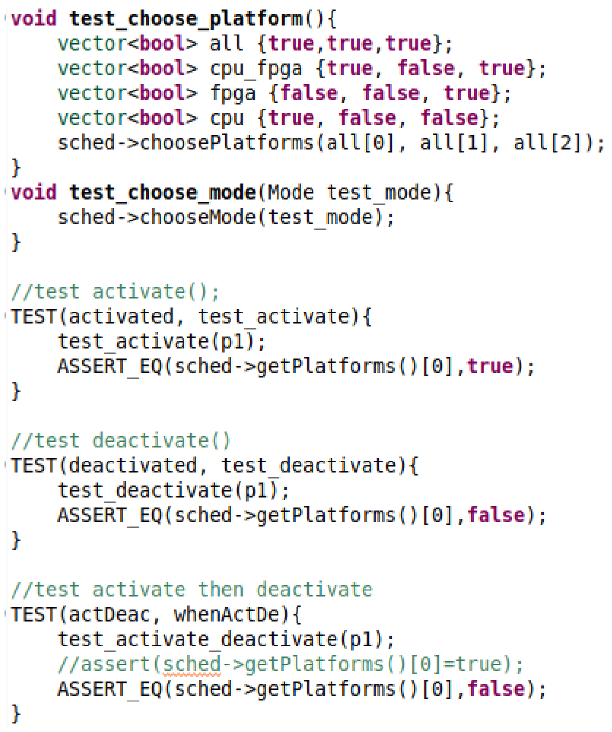
The scheduler is responsible of providing the system with information about platforms it can run on, depending on the user's choise of platforms and operation mode. See 10



```
TEST(write_result, test_write_result){
    Image* dog= new Image(227,227,3,1);
    DataSaver* data = new DataSaver();
    auto r = make_unique<Result>();
    r->save_result("lion", 90.66);
    data->set_result(dog->id, move(r));
    data->write_result_in_file(dog->id);
    std::ifstream infile(path::resultpath);
    bool exist = infile.good();
    ASSERT_EQ(exist,true);
}
TEST(aggregate, test_aggregate){
    Image* i1= new Image(227,227,3,1);
    Image* i2=new Image(227,227,3,1);
    Image* i3=new Image(227,227,3,1);
    auto r1= make_unique<Result>();
    auto r2= make_unique<Result>();
    auto r3= make_unique<Result>();
    DataSaver* save = new DataSaver();
    r1->save_result("dog",90);
    r1->save_result("cat",50);
    r1->save_result("truck",20);
    save->set_result(i1->id,move(r1));
    r2->save_result("dog", 80);
    r2->save_result("truck",60);
    r2->save_result("cat", 10);
    save->set_result(i2->id,move(r2));
    r3->save_result("dog", 70);
    r3->save_result("fish", 15);
    r3->save_result("truck", 10);
    save->set_result(i3->id,move(r3));
    auto final= save->aggregate();
```

Figure 9: data saver test

3.3 Test results



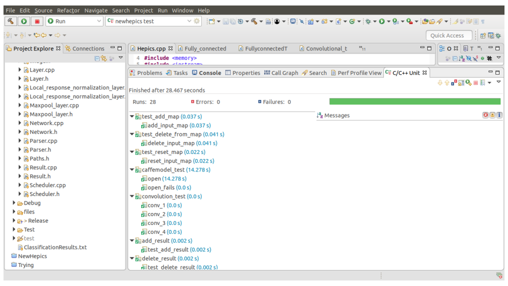
```
void test_choose_platform(){
    vector<bool> all {true,true,true};
    vector<bool> cpu_fpga {true, false, true};
    vector<bool> fpga {false, false, true};
    vector<bool> cpu {true, false, false};
    sched->choosePlatforms(all[0], all[1], all[2]);
}
void test_choose_mode(Mode test_mode){
    sched->chooseMode(test_mode);
}

//test activate();
TEST(activated, test_activate){
    test_activate(p1);
    ASSERT_EQ(sched->getPlatforms()[0],true);
}

//test deactivate()
TEST(deactivated, test_deactivate){
    test_deactivate(p1);
    ASSERT_EQ(sched->getPlatforms()[0],false);
}

//test activate then deactivate
TEST(actDeac, whenActDe){
    test_activate_deactivate(p1);
    //assert(sched->getPlatforms()[0]=true);
    ASSERT_EQ(sched->getPlatforms()[0],false);
}
```

Figure 10: scheduler test

4 Coverage

Gcov is a test coverage program. We used it in concert with GCC to analyze our programs and help create more efficient, faster running code, and discover untested parts of the program. Gcov is an efficient profiling tool to help discover where the optimization efforts will best affect the code.
Gcov profiling tools helped us analyze our code's performance. By using gcov we found some basic performance statistics, such as:

- how often each line of code executes

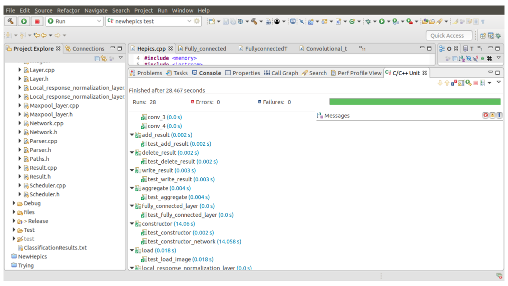


```
4 #include <memory>
```

Layer.cpp
Layer.h
Local_response_normalization_layer.
Local_response_normalization_layer.
Maxpool_layer.cpp
Maxpool_layer.h
Network.cpp
Network.h
Parser.cpp
Parser.h
Paths.h
Result.cpp
Result.h
Scheduler.cpp
Scheduler.h
Debug
files
> Release
Test
test
ClassificationResults.txt
NewHepics
Trying

```
Finished after 28.467 seconds
Runs: 28        Errors: 0        Failures: 0

test_add_map (0.037 s)
    add_input_map (0.037 s)
test_delete_from_map (0.041 s)
    delete_input_map (0.041 s)
test_reset_map (0.022 s)
    reset_input_map (0.022 s)
caffemodel_test (14.278 s)
    open (14.278 s)
    open_fails (0.0 s)
convolution_test (0.0 s)
    conv_1 (0.0 s)
    conv_2 (0.0 s)
    conv_3 (0.0 s)
    conv_4 (0.0 s)
add_result (0.002 s)
    test_add_result (0.002 s)
delete_result (0.002 s)
    test_delete_result (0.002 s)
```

- what lines of code are actually executed
- how much computing time each section of code uses

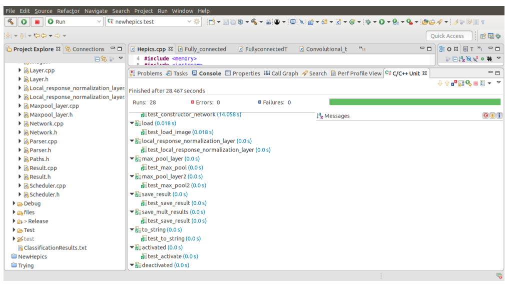


```
4 #include <memory>
```

```
Finished after 28.467 seconds
Runs: 28        Errors: 0        Failures: 0

    conv_3 (0.0 s)
    conv_4 (0.0 s)
▼ add_result (0.002 s)
    test_add_result (0.002 s)
▼ delete_result (0.002 s)
    test_delete_result (0.002 s)
▼ write_result (0.003 s)
    test_write_result (0.003 s)
▼ aggregate (0.004 s)
    test_aggregate (0.004 s)
▼ fully_connected_layer (0.0 s)
    test_fully_connected_layer (0.0 s)
▼ constructor (14.06 s)
    test_constructor (0.002 s)
    test_constructor_network (14.058 s)
▼ load (0.018 s)
    test_load_image (0.018 s)
▼ local_response_normalization_layer (0.0 s)
```

5 Current Overall State

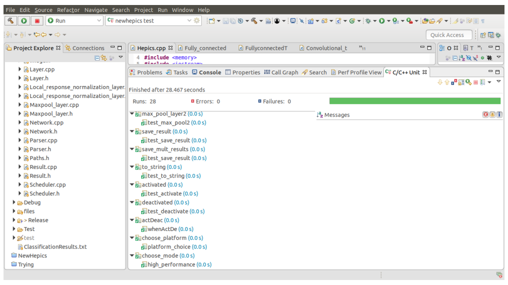
Eclipse CDT , which was used to manage the back-end build, and QtCreator ,which was used to manage the GUI build, are not especially set to work with each other, and use two different ways of building and compiling projects. Since the classification is the core of this project, we opted to run both seperately until we manage to make both of the GUI and classification run on eclipse. We trust we have made a big step in this direction, since basic GUI's are now running on eclipse, however we still have compiling errors when merging both of the main projects.



```
4 #include <memory>
```

```
Layer.cpp
Layer.h
Local_response_normalization_layer.
Local_response_normalization_layer.
Maxpool_layer.cpp
Maxpool_layer.h
Network.cpp
Network.h
Parser.cpp
Parser.h
Paths.h
Result.cpp
Result.h
Scheduler.cpp
Scheduler.h
Debug
files
> Release
Test
test
ClassificationResults.txt
NewHepics
Trying
```

```
Finished after 28.467 seconds
Runs: 28        Errors: 0        Failures: 0
  test_constructor_network (14.058 s)
load (0.018 s)
  test_load_image (0.018 s)
local_response_normalization_layer (0.0 s)
  test_local_response_normalization_layer (0.0 s)
max_pool_layer (0.0 s)
  test_max_pool (0.0 s)
max_pool_layer2 (0.0 s)
  test_max_pool2 (0.0 s)
save_result (0.0 s)
  test_save_result (0.0 s)
save_mult_results (0.0 s)
  test_save_result (0.0 s)
to_string (0.0 s)
  test_to_string (0.0 s)
activated (0.0 s)
  test_activate (0.0 s)
deactivated (0.0 s)
```



```
Layer.cpp
Layer.h
Local_response_normalization_layer.
Local_response_normalization_layer.
Maxpool_layer.cpp
Maxpool_layer.h
Network.cpp
Network.h
Parser.cpp
Parser.h
Paths.h
Result.cpp
Result.h
Scheduler.cpp
Scheduler.h
Debug
files
> Release
Test
test
ClassificationResults.txt
NewHepics
Trying
```

```
4 #include <memory>
```

```
Finished after 28.467 seconds
Runs: 28        Errors: 0        Failures: 0
max_pool_layer2 (0.0 s)
    test_max_pool2 (0.0 s)
save_result (0.0 s)
    test_save_result (0.0 s)
save_mult_results (0.0 s)
    test_save_result (0.0 s)
to_string (0.0 s)
    test_to_string (0.0 s)
activated (0.0 s)
    test_activate (0.0 s)
deactivated (0.0 s)
    test_deactivate (0.0 s)
actDeac (0.0 s)
    whenActDe (0.0 s)
choose_platform (0.0 s)
    platform_choice (0.0 s)
choose_mode (0.0 s)
    high_performance (0.0 s)
```