Sign up

Xilinx / **Vitis-AI**

Code | Issues 122 | Pull requests 18 | Actions | Projects | Security

master

Vitis-AI / demo / VART / resnet50 / src / **main.cc**

hanxue Vitis-AI 1.3.1 Release (#318)

History

1 contributor

Raw | Blame

Executable File | 305 lines (271 sloc) | 8.54 KB

```
/*
 * Copyright 2019 Xilinx Inc.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#include <assert.h>
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <unistd.h>

#include <cassert>
#include <cmath>
#include <cstdio>
```

```cpp
#include <fstream>
#include <iomanip>
#include <iostream>
#include <queue>
#include <string>
#include <vector>

#include "common.h"
/* header file OpenCV for image processing */
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

GraphInfo shapes;

const string baseImagePath = "../images/";
const string wordsPath = "./";

/**
 * @brief put image names to a vector
 *
 * @param path - path of the image direcotry
 * @param images - the vector of image name
 *
 * @return none
 */
void ListImages(string const& path, vector<string>& images) {
  images.clear();
  struct dirent* entry;

  /*Check if path is a valid directory path. */
  struct stat s;
  lstat(path.c_str(), &s);
  if (!S_ISDIR(s.st_mode)) {
    fprintf(stderr, "Error: %s is not a valid directory!\n", path.c_str());
    exit(1);
  }

  DIR* dir = opendir(path.c_str());
  if (dir == nullptr) {
    fprintf(stderr, "Error: Open %s path failed.\n", path.c_str());
    exit(1);
  }

  while ((entry = readdir(dir)) != nullptr) {
    if (entry->d_type == DT_REG || entry->d_type == DT_UNKNOWN) {
      string name = entry->d_name;
      string ext = name.substr(name.find_last_of(".") + 1);
      if ((ext == "JPEG") || (ext == "jpeg") || (ext == "JPG") ||
          (ext == "jpg") || (ext == "PNG") || (ext == "png")) {
```

```
        images.push_back(name);
      }
    }
  }

  closedir(dir);
}

/**
 * @brief load kinds from file to a vector
 *
 * @param path - path of the kinds file
 * @param kinds - the vector of kinds string
 *
 * @return none
 */
void LoadWords(string const& path, vector<string>& kinds) {
  kinds.clear();
  ifstream fkinds(path);
  if (fkinds.fail()) {
    fprintf(stderr, "Error : Open %s failed.\n", path.c_str());
    exit(1);
  }
  string kind;
  while (getline(fkinds, kind)) {
    kinds.push_back(kind);
  }

  fkinds.close();
}

/**
 * @brief calculate softmax
 *
 * @param data - pointer to input buffer
 * @param size - size of input buffer
 * @param result - calculation result
 *
 * @return none
 */
void CPUCalcSoftmax(const float* data, size_t size, float* result) {
  assert(data && result);
  double sum = 0.0f;

  for (size_t i = 0; i < size; i++) {
    result[i] = exp(data[i]);
    sum += result[i];
  }
  for (size_t i = 0; i < size; i++) {
    result[i] /= sum;
  }
```

```cpp
}

/**
 * @brief Get top k results according to its probability
 *
 * @param d - pointer to input data
 * @param size - size of input data
 * @param k - calculation result
 * @param vkinds - vector of kinds
 *
 * @return none
 */
void TopK(const float* d, int size, int k, vector<string>& vkinds) {
  assert(d && size > 0 && k > 0);
  priority_queue<pair<float, int>> q;

  for (auto i = 0; i < size; ++i) {
    q.push(pair<float, int>(d[i], i));
  }

  for (auto i = 0; i < k; ++i) {
    pair<float, int> ki = q.top();
    printf("top[%d] prob = %-8f  name = %s\n", i, d[ki.second],
           vkinds[ki.second].c_str());
    q.pop();
  }
}

/**
 * @brief Run DPU Task for ResNet50
 *
 * @param taskResnet50 - pointer to ResNet50 Task
 *
 * @return none
 */
void runResnet50(vart::Runner* runner) {
  /* Mean value for ResNet50 specified in Caffe prototxt */
  vector<string> kinds, images;

  /* Load all image names.*/
  ListImages(baseImagePath, images);
  if (images.size() == 0) {
    cerr << "\nError: No images existing under " << baseImagePath << endl;
    return;
  }

  /* Load all kinds words.*/
  LoadWords(wordsPath + "words.txt", kinds);
  if (kinds.size() == 0) {
    cerr << "\nError: No words exist in file words.txt." << endl;
    return;
  }
  float mean[3] = {104, 107, 123};

  /* get in/out tensors and dims*/
  auto outputTensors = runner->get_output_tensors();
  auto inputTensors = runner->get_input_tensors();
  auto out_dims = outputTensors[0]->get_shape();
  auto in_dims = inputTensors[0]->get_shape();

  /*get shape info*/
  int outSize = shapes.outTensorList[0].size;
  int inSize = shapes.inTensorList[0].size;
  int inHeight = shapes.inTensorList[0].height;
  int inWidth = shapes.inTensorList[0].width;

  int batchSize = in_dims[0];

  std::vector<std::unique_ptr<vart::TensorBuffer>> inputs, outputs;

  vector<Mat> imageList;
  float* imageInputs = new float[inSize * batchSize];

  float* softmax = new float[outSize];
  float* FCResult = new float[batchSize * outSize];
  std::vector<vart::TensorBuffer*> inputsPtr, outputsPtr;
  std::vector<std::shared_ptr<xir::Tensor>> batchTensors;
  /*run with batch*/
  for (unsigned int n = 0; n < images.size(); n += batchSize) {
    unsigned int runSize =
        (images.size() < (n + batchSize)) ? (images.size() - n) : batchSize;
    in_dims[0] = runSize;
    out_dims[0] = batchSize;
    for (unsigned int i = 0; i < runSize; i++) {
      Mat image = imread(baseImagePath + images[n + i]);

      /*image pre-process*/
      Mat image2 = cv::Mat(inHeight, inWidth, CV_8SC3);
      resize(image, image2, Size(inHeight, inWidth), 0, 0);
      for (int h = 0; h < inHeight; h++) {
        for (int w = 0; w < inWidth; w++) {
          for (int c = 0; c < 3; c++) {
            imageInputs[i * inSize + h * inWidth * 3 + w * 3 + c] =
                image2.at<Vec3b>(h, w)[c] - mean[c];
          }
        }
      }
      imageList.push_back(image);
    }

    /* in/out tensor refactory for batch inout/output */
    batchTensors.push_back(std::shared_ptr<xir::Tensor>(xir::Tensor::create(
        inputTensors[0]->get_name(), in_dims,
        xir::DataType{xir::DataType::FLOAT, sizeof(float) * 8u})));
    inputs.push_back(std::make_unique<CpuFlatTensorBuffer>(
        imageInputs, batchTensors.back().get()));
    batchTensors.push_back(std::shared_ptr<xir::Tensor>(xir::Tensor::create(
        outputTensors[0]->get_name(), out_dims,
        xir::DataType{xir::DataType::FLOAT, sizeof(float) * 8u})));
    outputs.push_back(std::make_unique<CpuFlatTensorBuffer>(
        FCResult, batchTensors.back().get()));

    /*tensor buffer input/output */
    inputsPtr.clear();
    outputsPtr.clear();
    inputsPtr.push_back(inputs[0].get());
    outputsPtr.push_back(outputs[0].get());

    /*run*/
    auto job_id = runner->execute_async(inputsPtr, outputsPtr);
    runner->wait(job_id.first, -1);
    for (unsigned int i = 0; i < runSize; i++) {
      cout << "\nImage : " << images[n + i] << endl;
      /* Calculate softmax on CPU and display TOP-5 classification results */
      CPUCalcSoftmax(&FCResult[i * outSize], outSize, softmax);
      TopK(softmax, outSize, 5, kinds);
      /* Display the impage */
      cv::imshow("Classification of ResNet50", imageList[i]);
      cv::waitKey(10000);
    }
    imageList.clear();
    inputs.clear();
    outputs.clear();
  }
  delete[] FCResult;
  delete[] imageInputs;
  delete[] softmax;
}

/**
 * @brief Entry for runing ResNet50 neural network
 *
 * @note Runner APIs prefixed with "dpu" are used to easily program &
 *       deploy ResNet50 on DPU platform.
 *
 */
int main(int argc, char* argv[]) {
  // Check args
  if (argc != 2) {
    cout << "Usage of resnet50 demo: ./resnet50 [model_file]" << endl;
    return -1;
  }
  auto graph = xir::Graph::deserialize(argv[1]);
  auto subgraph = get_dpu_subgraph(graph.get());
  CHECK_EQ(subgraph.size(), 1u)
      << "resnet50 should have one and only one dpu subgraph.";
  LOG(INFO) << "create running for subgraph: " << subgraph[0]->get_name();
  /*create runner*/
  auto runner = vart::Runner::create_runner(subgraph[0], "run");
  // ai::XdpuRunner* runner = new ai::XdpuRunner("./");
  /*get in/out tensor*/
  auto inputTensors = runner->get_input_tensors();
  auto outputTensors = runner->get_output_tensors();

  /*get in/out tensor shape*/
  int inputCnt = inputTensors.size();
  int outputCnt = outputTensors.size();
  TensorShape inshapes[inputCnt];
  TensorShape outshapes[outputCnt];
  shapes.inTensorList = inshapes;
  shapes.outTensorList = outshapes;
  getTensorShape(runner.get(), &shapes, inputCnt, outputCnt);

  /*run with batch*/
  runResnet50(runner.get());
  return 0;
}
```