Daisykit

Release 0.2.0.1

Daisykit Authors

May 05, 2022

PYTHON TUTORIALS

CHAPTER

ONE

DAISYKIT PYTHON

https://pypi.org/project/daisykit/

Daisykit is an easy AI toolkit for software engineers to integrate pretrained AI models and pipelines into their projects. You DON'T need to be an AI engineer to build AI software. This open source project includes:

- **Daisykit SDK - C++**, the core of models and algorithms in NCNN deep learning framework.
- **Daisykit Python** wrapper for easy integration with Python.
- **Daisykit Android** - Example app demonstrate how to use Daisykit SDK in Android.

1.1 1. Installation

For Windows:

```
pip3 install daisykit
```

For Ubuntu:

- Install dependencies

```
sudo apt install pybind11-dev # Pybind11 - For Python/C++ Wrapper
sudo apt install libopencv-dev # For OpenCV
sudo apt install libvulkan-dev # Optional - For GPU support
```

- Install DaisyKit (compile from source)

```
pip3 install --upgrade pip # Ensure pip is updated
pip3 install daisykit
```

For other platforms:

- Install OpenCV, Pybind11 and Vulkan development package (if you want GPU support)
- Install DaisyKit (compile from source)

```
pip3 install --upgrade pip # Ensure pip is updated
pip3 install daisykit
```

1.2 2. Note for Python build

Current CD (continuous delivery) flow is partial, which means we only have prebuilt linux wheels for x86_64 and for Windows.

- Prebuilt wheels for linux x86_64 are built with Github actions.
- Windows wheels (64bit) are built manually on a local machine.
- macOS prebuilt wheels are not available for now. However, you can install dependencies (OpenCV, Vulkan) manually, then install Daisykit with pip command.

We will be happy if you can make a pull request to make the CD build fully automated. A good choice is using Github flow for all building tasks.

Current steps for Windows build:

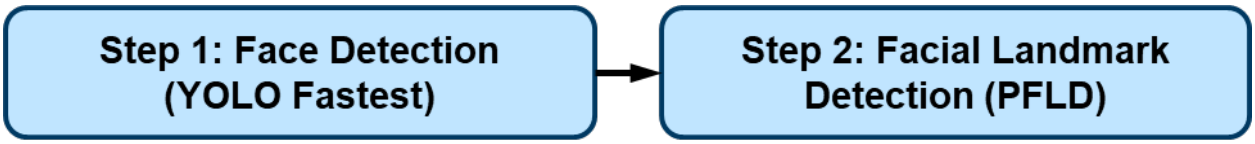
```
bash ./build_tools/py_windows/build_dk_all_pythons_windows.sh
bash ./build_tools/py_windows/build_dk_python_source_dist.sh
bash ./build_tools/upload_pypi.sh
```

CHAPTER

TWO

PYTHON: FACE DETECTION

Face detector flow in Daisykit contains a face detection model based on *YOLO Fastest* and a facial landmark detection model based on *PFLD*. In addition, to encourage makers to join hands in the fighting with COVID-19, we selected a face detection model that can recognize people wearing face masks or not.



Let's see into the source code below to understand how to run this model with your webcam. First, we initialize the flow with a `config` dictionary. It contains information about the models used in the flow. `get_asset_file()` function will automatically download the models and weights files from *https://github.com/DaisyLabSolutions/daisykit-assets*, so you don't have to care about downloading them manually. The downloading only happens the first time we use the models. After that, you can run this code offline. You also can download all files yourself and put the paths to file in instead of `get_asset_file(<assets address>)`. If you don't need facial landmark output, set `with_landmark` to `False`.

```
import cv2
import json
from daisykit.utils import get_asset_file, to_py_type
import daisykit

config = {
    "face_detection_model": {
        "model": get_asset_file("models/face_detection/yolo_fastest_with_mask/yolo-
↪fastest-opt.param"),
        "weights": get_asset_file("models/face_detection/yolo_fastest_with_mask/yolo-
↪fastest-opt.bin"),
        "input_width": 320,
        "input_height": 320,
        "score_threshold": 0.7,
        "iou_threshold": 0.5,
        "use_gpu": False
    },
    "with_landmark": True,
    "facial_landmark_model": {
        "model": get_asset_file("models/facial_landmark/pfld-sim.param"),
        "weights": get_asset_file("models/facial_landmark/pfld-sim.bin"),
        "input_width": 112,
```

(continues on next page)

(continued from previous page)

```
        "input_height": 112,
        "use_gpu": False
    }
}

face_detector_flow = daisykit.FaceDetectorFlow(json.dumps(config))

# Open video stream from webcam
vid = cv2.VideoCapture(0)

while(True):

    # Capture the video frame
    ret, frame = vid.read()

    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    faces = face_detector_flow.Process(frame)
    # for face in faces:
    #     print([face.x, face.y, face.w, face.h,
    #            face.confidence, face.wearing_mask_prob])
    face_detector_flow.DrawResult(frame, faces)

    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

    # Convert faces to Python list of dict
    faces = to_py_type(faces)

    # Display the resulting frame
    cv2.imshow('frame', frame)

    # The 'q' button is set as the
    # quitting button you may use any
    # desired button of your choice
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# After the loop release the cap object
vid.release()
# Destroy all the windows
cv2.destroyAllWindows()
```

All flows in Daisykit are initialized by a configuration string. Therefore we use **json.dumps()** to convert the config dictionary to a string. For example, in face detector flow:

```
face_detector_flow = daisykit.FaceDetectorFlow(json.dumps(config))
```

Run the flow to get detected faces by:

```
faces = face_detector_flow.Process(frame)
```

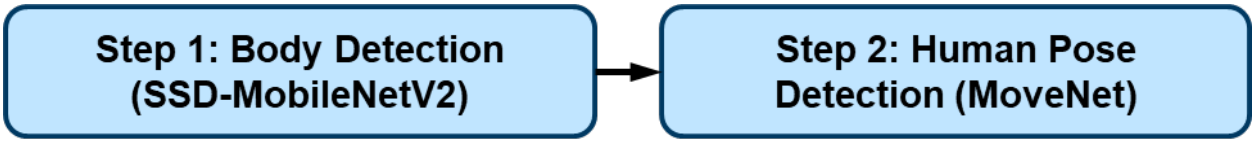
You can use the **DrawResult()** method to visualize the result or write a drawing function yourself. This AI flow can be used in DIY projects such as *smart COVID-19 camera* or Snap-chat-like camera decorators.

CHAPTER

THREE

PYTHON: HUMAN POSE

The human pose detector module contains an SSD-MobileNetV2 body detector and a ported Google MoveNet model for human keypoints. This module can be applied in fitness applications and AR games.



Source code:

```
import cv2
import json
from daisykit.utils import get_asset_file, to_py_type
from daisykit import HumanPoseMoveNetFlow

config = {
    "person_detection_model": {
        "model": get_asset_file("models/human_detection/ssd_mobilenetv2.param"),
        "weights": get_asset_file("models/human_detection/ssd_mobilenetv2.bin"),
        "input_width": 320,
        "input_height": 320,
        "use_gpu": False
    },
    "human_pose_model": {
        "model": get_asset_file("models/human_pose_detection/movenet/lightning.param"),
        "weights": get_asset_file("models/human_pose_detection/movenet/lightning.bin"),
        "input_width": 192,
        "input_height": 192,
        "use_gpu": False
    }
}

human_pose_flow = HumanPoseMoveNetFlow(json.dumps(config))

# Open video stream from webcam
vid = cv2.VideoCapture(0)

while(True):
```

(continues on next page)

(continued from previous page)

```
    # Capture the video frame
    ret, frame = vid.read()

    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    poses = human_pose_flow.Process(frame)
    human_pose_flow.DrawResult(frame, poses)

    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

    # Convert poses to Python list of dict
    poses = to_py_type(poses)

    # Display the result frame
    cv2.imshow('frame', frame)

    # Press 'q' to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

CHAPTER

FOUR

PYTHON: BACKGROUND MATTING

Background matting use only one segmentation model to generate a human body mask. This mask can figure out which pixels belong to humans and which belong to the background. This output can be used for background replacement (like in the Google Meet app). The segmentation model was taken from *this implementation* by *nihui*, the author of the NCNN framework. The author also has a webpage for a live demo on web browsers.

https://github.com/nihui/ncnn-webassembly-portrait-segmentation.

Source code:

```
import cv2
import json
from daisykit.utils import get_asset_file
from daisykit import BackgroundMattingFlow

config = {
    "background_matting_model": {
        "model": get_asset_file("models/background_matting/erd/erdnet.param"),
        "weights": get_asset_file("models/background_matting/erd/erdnet.bin"),
        "input_width": 256,
        "input_height": 256,
        "use_gpu": False
    }
}

# Load background
default_bg_file = get_asset_file("images/background.jpg")
background = cv2.imread(default_bg_file)
background = cv2.cvtColor(background, cv2.COLOR_BGR2RGB)

background_matting_flow = BackgroundMattingFlow(json.dumps(config), background)

# Open video stream from webcam
vid = cv2.VideoCapture(0)

while(True):

    # Capture the video frame
    ret, frame = vid.read()

    image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

(continues on next page)

(continued from previous page)

```
    mask = background_matting_flow.Process(image)
    background_matting_flow.DrawResult(image, mask)

    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    # Display the result frame
    cv2.imshow('frame', frame)
    cv2.imshow('result', image)

    # Press 'q' to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

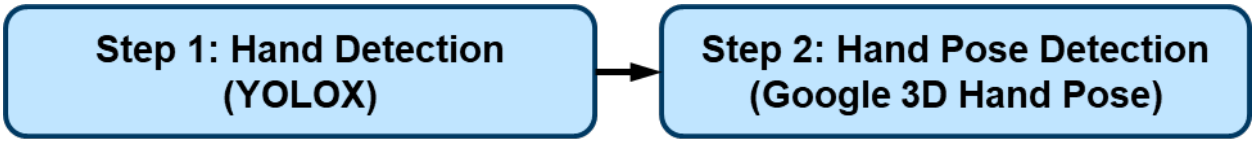
In the source code, we use `get_asset_file("images/background.jpg")` to download the default background. You can use another image for the background by replacing it with a path to an image file.

CHAPTER

FIVE

PYTHON: HAND POSE DETECTION

The hand pose detection flow comprises two models: a hand detection model based on YOLOX and a 3D hand pose detection model released by Google this November. Thanks to *FeiGeChuanShu* for the effort in early model conversion.



This hand pose flow can be used in AR games, hand gesture control, and many cool DIY projects.

Source code:

```
import cv2
import json
from daisykit.utils import get_asset_file, to_py_type
from daisykit import HandPoseDetectorFlow

config = {
    "hand_detection_model": {
        "model": get_asset_file("models/hand_pose/yolox_hand_swish.param"),
        "weights": get_asset_file("models/hand_pose/yolox_hand_swish.bin"),
        "input_width": 256,
        "input_height": 256,
        "score_threshold": 0.45,
        "iou_threshold": 0.65,
        "use_gpu": False
    },
    "hand_pose_model": {
        "model": get_asset_file("models/hand_pose/hand_lite-op.param"),
        "weights": get_asset_file("models/hand_pose/hand_lite-op.bin"),
        "input_size": 224,
        "use_gpu": False
    }
}

flow = HandPoseDetectorFlow(json.dumps(config))

# Open video stream from webcam
vid = cv2.VideoCapture(0)
```

(continues on next page)

(continued from previous page)

```
while(True):

    # Capture the video frame
    ret, frame = vid.read()

    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    poses = flow.Process(frame)
    flow.DrawResult(frame, poses)

    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

    # Convert poses to Python list of dict
    poses = to_py_type(poses)

    # Display the result frame
    cv2.imshow('frame', frame)

    # Press 'q' to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

In the above source code, `input_width` and `input_height` of the `hand_detection_model` can be adjusted for speed/accuracy trade-off.

CHAPTER

SIX

PYTHON: BARCODE DETECTION

Barcodes can be used in a wide range of robotics and software applications. That's why we integrated a barcode reader into Daisykit. The core algorithms of the barcode reader are from *the Zxing-CPP project*. This barcode processor can read QR codes and bar codes in different formats.

Source code:

```
import cv2
import json
from daisykit.utils import get_asset_file
from daisykit import BarcodeScannerFlow

config = {
    "try_harder": True,
    "try_rotate": True
}

barcode_scanner_flow = BarcodeScannerFlow(json.dumps(config))

# Open video stream from webcam
vid = cv2.VideoCapture(0)

while(True):

    # Capture the video frame
    ret, frame = vid.read()

    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    result = barcode_scanner_flow.Process(frame, draw=True)

    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

    # Display the result frame
    cv2.imshow('frame', frame)

    # Press 'q' to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

CHAPTER

SEVEN

PYTHON: OBJECT DETECTION

A general-purpose object detector based on *YOLOX* is integrated with Daisykit. The models are trained on the COCO dataset using the *official repository of YOLOX*. You can retrain the model with your custom dataset and convert it to NCNN format, which can be integrated into Daisykit easily.

Source code:

```
import cv2
import json
from daisykit.utils import get_asset_file, to_py_type
from daisykit import ObjectDetectorFlow

config = {
    "object_detection_model": {
        "model": get_asset_file("models/object_detection/yolox-tiny.param"),
        "weights": get_asset_file("models/object_detection/yolox-tiny.bin"),
        "input_width": 416,
        "input_height": 416,
        "score_threshold": 0.5,
        "iou_threshold": 0.8,
        "use_gpu": False,
        "class_names": [
            "person", "bicycle", "car", "motorcycle", "airplane", "bus", "train", "truck
↪", "boat", "traffic light",
            "fire hydrant", "stop sign", "parking meter", "bench", "bird", "cat", "dog",
↪"horse", "sheep", "cow",
            "elephant", "bear", "zebra", "giraffe", "backpack", "umbrella", "handbag",
↪"tie", "suitcase", "frisbee",
            "skis", "snowboard", "sports ball", "kite", "baseball bat", "baseball glove",
↪ "skateboard", "surfboard",
            "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon",
↪"bowl", "banana", "apple",
            "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut",
↪"cake", "chair", "couch",
            "potted plant", "bed", "dining table", "toilet", "tv", "laptop", "mouse",
↪"remote", "keyboard", "cell phone",
            "microwave", "oven", "toaster", "sink", "refrigerator", "book", "clock",
↪"vase", "scissors", "teddy bear",
            "hair drier", "toothbrush"
        ]
```

(continues on next page)

(continued from previous page)

```
    }
}

flow = ObjectDetectorFlow(json.dumps(config))

# Open video stream from webcam
vid = cv2.VideoCapture(0)

while(True):

    # Capture the video frame
    ret, frame = vid.read()

    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    poses = flow.Process(frame)
    flow.DrawResult(frame, poses)

    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

    # Convert poses to Python list of dict
    poses = to_py_type(poses)

    # Display the result frame
    cv2.imshow('frame', frame)

    # Press 'q' to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

If you have any difficulty running the examples above, we prepared a Colab environment to try them without setting up a local environment.

Access our Colab notebook at: *https://colab.research.google.com/drive/1LFg3xcoFr3wxuJmn3c4LEJiW2G7oP7F5#scrollTo=2BLn9Ofa*

CHAPTER

EIGHT

DAISYKIT C++

Daisykit SDK - C++ is the core of models and algorithms in NCNN deep learning framework. Using C++ code provides often provides the best performance for the algorithms.

8.1 1. Environment Setup

8.1.1 Ubuntu

Install packages from Terminal

```
sudo apt install -y build-essential libopencv-dev
sudo apt install -y libvulkan-dev vulkan-utils
sudo apt install -y mesa-vulkan-drivers # For Intel GPU support
```

8.1.2 Windows

For Windows, Visual Studio 2019 + Git Bash is recommended.

- Download and extract OpenCV from the official website, and add `OpenCV_DIR` to path.
- Download precompiled NCNN.

8.2 2. Build and run C++ examples

Clone the source code:

```
git clone https://github.com/DaisyLabSolutions/daisykit.git --recursive
cd daisykit
```

8.2.1 Ubuntu

Build Daisykit:

```
mkdir build
cd build
cmake .. -Dncnn_FIND_PATH="<path to ncnn lib>"
make
```

Run face detection example:

```
./bin/demo_face_detector_graph
```

If you dont specify ncnn_FIND_PATH, NCNN will be built from scratch.

8.2.2 Windows

Build Daisykit:

```
mkdir build
cd build
cmake -G "Visual Studio 16 2019" -Dncnn_FIND_PATH="<path to ncnn lib>" ..
cmake --build . --config Release
```

Run face detection example:

```
./bin/Release/demo_face_detector_graph
```

8.3 3. C++ Coding convention

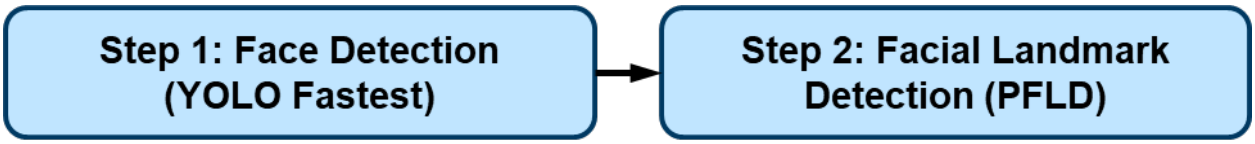
Read coding convention and contribution guidelines here.

CHAPTER

NINE

C++: FACE DETECTION

Face detector flow in Daisykit contains a face detection model based on *YOLO Fastest* and a facial landmark detection model based on *PFLD*. In addition, to encourage makers to join hands in the fighting with COVID-19, we selected a face detection model that can recognize people wearing face masks or not.



9.1 1. Sequential flow

Source code: `src/examples/demo_face_detector.cpp`.

```cpp
#include "daisykit/common/types.h"
#include "daisykit/flows/face_detector_flow.h"
#include "third_party/json.hpp"

#include <stdio.h>
#include <fstream>
#include <iostream>
#include <opencv2/opencv.hpp>
#include <streambuf>
#include <string>

using namespace cv;
using namespace std;
using json = nlohmann::json;
using namespace daisykit;
using namespace daisykit::flows;

int main(int, char**) {
  std::ifstream t("configs/face_detector_config.json");
  std::string config_str((std::istreambuf_iterator<char>(t)),
                         std::istreambuf_iterator<char>());

  FaceDetectorFlow flow(config_str, true);
```

(continues on next page)

(continued from previous page)

```
  Mat frame;
  VideoCapture cap(0);

  while (1) {
    cap >> frame;
    cv::Mat rgb;
    cv::cvtColor(frame, rgb, cv::COLOR_BGR2RGB);

    std::vector<types::Face> faces = flow.Process(rgb);
    flow.DrawResult(rgb, faces);

    cv::Mat draw;
    cv::cvtColor(rgb, draw, cv::COLOR_RGB2BGR);
    imshow("Image", draw);
    waitKey(1);
  }

  return 0;
}
```

Update the configurations by modifying config files in `assets/configs`.

9.2 2. Multithreading mode with graph

In order to use multithreading mode (graph mode) to improve the FPS, try the example in `src/examples/demo_face_detector_graph.cpp`.

```
#include "daisykit/graphs/core/graph.h"
#include "daisykit/graphs/core/node.h"
#include "daisykit/graphs/nodes/models/face_detector_node.h"
#include "daisykit/graphs/nodes/models/face_landmark_detector_node.h"
#include "daisykit/graphs/nodes/packet_distributor_node.h"
#include "daisykit/graphs/nodes/visualizers/face_visualizer_node.h"
#include "third_party/json.hpp"

#include <stdio.h>
#include <fstream>
#include <iostream>
#include <opencv2/opencv.hpp>
#include <streambuf>
#include <string>

using namespace cv;
using namespace std;
using namespace daisykit;
using namespace daisykit::graphs;

int main(int, char**) {
  // Create processing nodes
  std::shared_ptr<nodes::PacketDistributorNode> packet_distributor_node =
```

(continues on next page)

(continued from previous page)

```
    std::make_shared<nodes::PacketDistributorNode>("packet_distributor",
                                                   NodeType::kAsyncNode);
std::shared_ptr<nodes::FaceDetectorNode> face_detector_node =
    std::make_shared<nodes::FaceDetectorNode>(
        "face_detector",
        "models/face_detection/yolo_fastest_with_mask/"
        "yolo-fastest-opt.param",
        "models/face_detection/yolo_fastest_with_mask/"
        "yolo-fastest-opt.bin",
        NodeType::kAsyncNode);
std::shared_ptr<nodes::FacialLandmarkDetectorNode>
    facial_landmark_detector_node =
        std::make_shared<nodes::FacialLandmarkDetectorNode>(
            "facial_landmark_detector",
            "models/facial_landmark/pfld-sim.param",
            "models/facial_landmark/pfld-sim.bin", NodeType::kAsyncNode);
std::shared_ptr<nodes::FaceVisualizerNode> face_visualizer_node =
    std::make_shared<nodes::FaceVisualizerNode>("face_visualizer",
                                                NodeType::kAsyncNode, true);

// Create connections between nodes
Graph::Connect(nullptr, "", packet_distributor_node.get(), "input",
               TransmissionProfile(2, true), true);

Graph::Connect(packet_distributor_node.get(), "output",
               face_detector_node.get(), "input",
               TransmissionProfile(2, true), true);

Graph::Connect(packet_distributor_node.get(), "output",
               facial_landmark_detector_node.get(), "image",
               TransmissionProfile(2, true), true);
Graph::Connect(face_detector_node.get(), "output",
               facial_landmark_detector_node.get(), "faces",
               TransmissionProfile(2, true), true);

Graph::Connect(packet_distributor_node.get(), "output",
               face_visualizer_node.get(), "image",
               TransmissionProfile(2, true), true);
Graph::Connect(facial_landmark_detector_node.get(), "output",
               face_visualizer_node.get(), "faces",
               TransmissionProfile(2, true), true);

// Need to init these nodes before use
// This method also start worker threads of asynchronous node
packet_distributor_node->Activate();
face_detector_node->Activate();
facial_landmark_detector_node->Activate();
face_visualizer_node->Activate();

VideoCapture cap(0);

while (1) {
```

(continues on next page)

(continued from previous page)

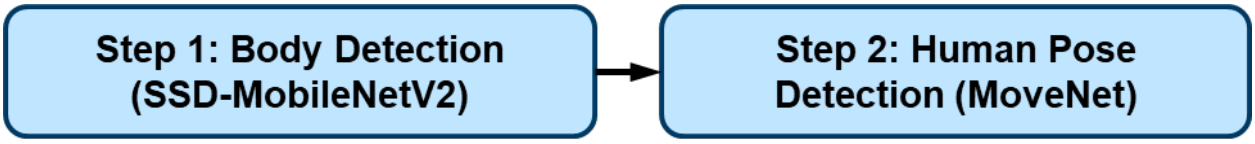
```
    Mat frame;
    cap >> frame;
    cv::cvtColor(frame, frame, cv::COLOR_BGR2RGB);
    std::shared_ptr<Packet> in_packet = Packet::MakePacket<cv::Mat>(frame);
    packet_distributor_node->Input("input", in_packet);
  }

  return 0;
}
```

CHAPTER

TEN

C++: HUMAN POSE

The human pose detector module contains an SSD-MobileNetV2 body detector and a ported Google MoveNet model for human keypoints. This module can be applied in fitness applications and AR games.



Source code: `src/examples/demo_movenet.cpp`.

```cpp
#include "daisykit/common/types.h"
#include "daisykit/flows/human_pose_movenet_flow.h"
#include "third_party/json.hpp"

#include <stdio.h>
#include <fstream>
#include <iostream>
#include <opencv2/opencv.hpp>
#include <streambuf>
#include <string>
#include <vector>

using namespace cv;
using namespace std;
using json = nlohmann::json;
using namespace daisykit::types;
using namespace daisykit::flows;

int main(int, char**) {
  std::ifstream t("configs/human_pose_movenet_config.json");
  std::string config_str((std::istreambuf_iterator<char>(t)),
                         std::istreambuf_iterator<char>());

  HumanPoseMoveNetFlow flow(config_str);

  Mat frame;
  VideoCapture cap(0);

  while (1) {
```

(continues on next page)

(continued from previous page)

```
    cap >> frame;
    cv::Mat rgb;
    cv::cvtColor(frame, rgb, cv::COLOR_BGR2RGB);

    std::vector<ObjectWithKeypoints> poses = flow.Process(rgb);
    flow.DrawResult(rgb, poses);

    cv::Mat draw;
    cv::cvtColor(rgb, draw, cv::COLOR_RGB2BGR);
    imshow("Image", draw);
    waitKey(1);
  }

  return 0;
}
```

Update the configurations by modifying config files in `assets/configs`.

CHAPTER

ELEVEN

C++: BACKGROUND MATTING

Background matting use only one segmentation model to generate a human body mask. This mask can figure out which pixels belong to humans and which belong to the background. This output can be used for background replacement (like in the Google Meet app). The segmentation model was taken from *this implementation* by *nihui*, the author of the NCNN framework. The author also has a webpage for a live demo on web browsers.

https://github.com/nihui/ncnn-webassembly-portrait-segmentation.

Source code: `src/examples/demo_background_matting.cpp`.

```
#include "daisykit/flows/background_matting_flow.h"
#include "third_party/json.hpp"

#include <stdio.h>
#include <fstream>
#include <iostream>
#include <opencv2/opencv.hpp>
#include <streambuf>
#include <string>

using namespace cv;
using namespace std;
using json = nlohmann::json;
using namespace daisykit::types;
using namespace daisykit::flows;

int main(int, char**) {
  std::ifstream t("configs/background_matting_config.json");
  std::string config_str((std::istreambuf_iterator<char>(t)),
                         std::istreambuf_iterator<char>());

  cv::Mat background = cv::imread("images/background.jpg");
  cv::cvtColor(background, background, cv::COLOR_BGR2RGB);
  BackgroundMattingFlow flow(config_str, background);

  Mat frame;
  VideoCapture cap(0);

  while (1) {
    cap >> frame;
    cv::Mat rgb;
```

(continues on next page)

(continued from previous page)

```
    cv::cvtColor(frame, rgb, cv::COLOR_BGR2RGB);
    cv::Mat draw = rgb.clone();

    cv::Mat mask = flow.Process(rgb);
    flow.DrawResult(draw, mask);

    cv::cvtColor(draw, draw, cv::COLOR_RGB2BGR);
    imshow("Image", draw);
    waitKey(1);
  }

  return 0;
}
```

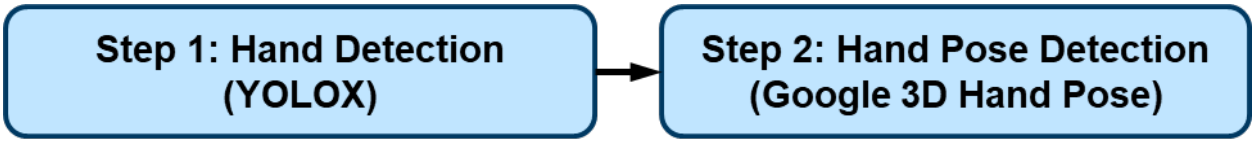
Update the configurations by modifying config files in `assets/configs`.

CHAPTER

TWELVE

C++: HAND POSE DETECTION

The hand pose detection flow comprises two models: a hand detection model based on YOLOX and a 3D hand pose detection model released by Google this November. Thanks to *FeiGeChuanShu* for the effort in early model conversion.



This hand pose flow can be used in AR games, hand gesture control, and many cool DIY projects.

Source code: `src/examples/demo_hand_pose_detector.cpp`.

```cpp
#include "daisykit/common/types.h"
#include "daisykit/flows/hand_pose_detector_flow.h"
#include "third_party/json.hpp"

#include <stdio.h>
#include <fstream>
#include <iostream>
#include <opencv2/opencv.hpp>
#include <streambuf>
#include <string>
#include <vector>

using namespace cv;
using namespace std;
using json = nlohmann::json;
using namespace daisykit::types;
using namespace daisykit::flows;

int main(int, char**) {
  std::ifstream t("configs/hand_pose_yolox_mp_config.json");
  std::string config_str((std::istreambuf_iterator<char>(t)),
                         std::istreambuf_iterator<char>());

  HandPoseDetectorFlow flow(config_str);

  Mat frame;
  VideoCapture cap(0);
```

(continues on next page)

(continued from previous page)

```
  while (1) {
    cap >> frame;
    cv::Mat rgb;
    cv::cvtColor(frame, rgb, cv::COLOR_BGR2RGB);

    std::vector<ObjectWithKeypointsXYZ> hands = flow.Process(rgb);
    flow.DrawResult(rgb, hands);

    cv::Mat draw;
    cv::cvtColor(rgb, draw, cv::COLOR_RGB2BGR);
    imshow("Image", draw);
    waitKey(1);
  }

  return 0;
}
```

Update the configurations by modifying config files in `assets/configs`. In the configuration file, `input_width` and `input_height` of the `hand_detection_model` can be adjusted for speed/accuracy trade-off.

CHAPTER

THIRTEEN

C++: BARCODE DETECTION

Barcodes can be used in a wide range of robotics and software applications. That's why we integrated a barcode reader into Daisykit. The core algorithms of the barcode reader are from *the Zxing-CPP project*. This barcode processor can read QR codes and bar codes in different formats.

Source code: `src/examples/demo_barcode_scanner.cpp`.

```cpp
#include "daisykit/flows/barcode_scanner_flow.h"
#include "third_party/json.hpp"

#include <stdio.h>
#include <fstream>
#include <iostream>
#include <opencv2/opencv.hpp>
#include <streambuf>
#include <string>

using namespace cv;
using namespace std;
using json = nlohmann::json;
using namespace daisykit::flows;

int main(int, char**) {
  std::ifstream t("configs/barcode_scanner_config.json");
  std::string config_str((std::istreambuf_iterator<char>(t)),
                         std::istreambuf_iterator<char>());

  BarcodeScannerFlow flow(config_str);

  Mat frame;
  VideoCapture cap(0);

  while (1) {
    cap >> frame;
    cv::Mat rgb;
    cv::cvtColor(frame, rgb, cv::COLOR_BGR2RGB);

    std::string result = flow.Process(rgb, true);
    if (!result.empty()) {
      std::cout << "New Scan Finished" << std::endl;
```

(continues on next page)

(continued from previous page)

```
      std::cout << result << std::endl;
    }

    cv::Mat draw;
    cv::cvtColor(rgb, draw, cv::COLOR_RGB2BGR);
    imshow("Image", draw);
    waitKey(1);
  }

  return 0;
}
```

Update the configurations by modifying config files in `assets/configs`.

CHAPTER

FOURTEEN

C++: OBJECT DETECTION

A general-purpose object detector based on *YOLOX* is integrated with Daisykit. The models are trained on the COCO dataset using the *official repository of YOLOX*. You can retrain the model with your custom dataset and convert it to NCNN format, which can be integrated into Daisykit easily.

Source code: `src/examples/demo_object_detector_yolox.cpp`.

```cpp
#include "daisykit/common/types.h"
#include "daisykit/flows/object_detector_flow.h"
#include "third_party/json.hpp"

#include <stdio.h>
#include <fstream>
#include <iostream>
#include <opencv2/opencv.hpp>
#include <streambuf>
#include <string>
#include <vector>

using namespace cv;
using namespace std;
using json = nlohmann::json;
using namespace daisykit::types;
using namespace daisykit::flows;

int main(int, char**) {
  std::ifstream t("configs/object_detector_yolox_config.json");
  std::string config_str((std::istreambuf_iterator<char>(t)),
                         std::istreambuf_iterator<char>());

  ObjectDetectorFlow flow(config_str);

  Mat frame;
  VideoCapture cap(0);

  while (1) {
    cap >> frame;
    cv::Mat rgb;
    cv::cvtColor(frame, rgb, cv::COLOR_BGR2RGB);
```

(continues on next page)

(continued from previous page)

```
    std::vector<Object> objects = flow.Process(rgb);
    flow.DrawResult(rgb, objects);

    cv::Mat draw;
    cv::cvtColor(rgb, draw, cv::COLOR_RGB2BGR);
    imshow("Image", draw);
    waitKey(1);
  }

  return 0;
}
```

Update the configurations by modifying config files in `assets/configs`.

CHAPTER

FIFTEEN

DAISYKIT ANDROID

Daisykit Android is built on top of Daisykit SDK. It contains wrapper code, examples to use Daisykit in Android mobile devices.

Github: https://github.com/DaisyLabSolutions/daisykit-android.

Very first Android demo:

15.1 I. Build and Run

Dependencies: Android NDK 21.3, CMake 3.10.2.

15.1.1 Step 1: Clone the source code

```
git clone --recurse-submodules https://github.com/DaisyLabSolutions/daisykit-android.git
```

Or

```
git clone https://github.com/DaisyLabSolutions/daisykit-android.git
cd daisykit-android
git submodule update --init
```

15.1.2 Step 2: Download NCNN

https://github.com/Tencent/ncnn/releases

- Download ncnn-YYYYMMDD-android-vulkan.zip or build ncnn for android yourself
- Extract ncnn-YYYYMMDD-android-vulkan.zip into **app/src/main/jni** and change the **ncnn_DIR** path to yours in **app/src/main/jni/CMakeLists.txt**

15.1.3 Step 3: Download OpenCV

https://github.com/nihui/opencv-mobile

- Download opencv-mobile-XYZ-android.zip
- Extract opencv-mobile-XYZ-android.zip into **app/src/main/jni** and change the **OpenCV_DIR** path to yours in **app/src/main/jni/CMakeLists.txt**

15.1.4 Step 4: Download assets

- Download all assets here and put into `app/src/main/assets/` folder.

15.1.5 Step 5: Build

- Open this project with Android Studio, build it and enjoy!

15.2 II. Some notes

- Android ndk camera is used for best efficiency.
- Crash may happen on very old devices for lacking HAL3 camera interface.
- All models are manually modified to accept dynamic input shape.
- Most small models run slower on GPU than on CPU, this is common.
- FPS may be lower in dark environment because of longer camera exposure time.

15.3 III. Errors and Fixes

15.3.1 1. Errors because of build tool version

```
No toolchains found in the NDK toolchains folder for ABI with prefix: arm-linux-
↪androideabi
```

```
clang ++: error: unknown argument: '-static-openmp'
```

How to fix? Install NDK 21.3, CMake 3.10.2.

15.3.2 2. Crashing app

Check if you have downloaded and extracted all assets. See into assets folder at `app/src/main/assets`. There should be 3 subfolders here: `models`, `configs`, and `images`.

CHAPTER

SIXTEEN

MODEL REFERENCES

Below are the resources for Daisykit deep learning models. At the current phase of Daisykit, we are focusing on model deployment and overall architecture. Therefore, source code and tutorial for training may be not available for now.

- **1. Person detection (person_detector) and Human pose (Ultralight-Nano-SimplePose):**
  - Pretrained models: https://github.com/dog-qiuqiu/Ultralight-SimplePose.
- **2. Facial landmark:**
  - Training code: https://github.com/polarisZhao/PFLD-pytorch.
  - Model conversion: https://github.com/nilseuropa/pfld_ncnn/.
- **3. Face detection (with wearing_mask output): WearMask**
  - Training & model conversion code: https://github.com/waittim/mask-detector.
- **4. Background matting:**
  - Pretrained model: https://github.com/nihui/ncnn-webassembly-portrait-segmentation.
- **5. Human pose estimation: Google MoveNet**
  - Original models: https://tfhub.dev/google/movenet/singlepose/lightning/4, https://tfhub.dev/google/movenet/singlepose/thunder/4, https://tfhub.dev/google/movenet/multipose/lightning/1.
  - Converted models from: https://github.com/FeiGeChuanShu/ncnn_Android_MoveNet.
- **6. Object detection model: YOLOX**
  - Training & model conversion code: https://github.com/Megvii-BaseDetection/YOLOX.
  - Converted by FeiGeChuanShu: https://github.com/FeiGeChuanShu/ncnn-android-yolox.

CHAPTER

SEVENTEEN

CONTRIBUTION GUIDELINES

17.1 I. Coding convention

17.1.1 Coding style setup

DaisyKit follows Google C++ Style Guide with some exceptions:

- Source code file names: `.cpp` for source file and `.h` for header files.

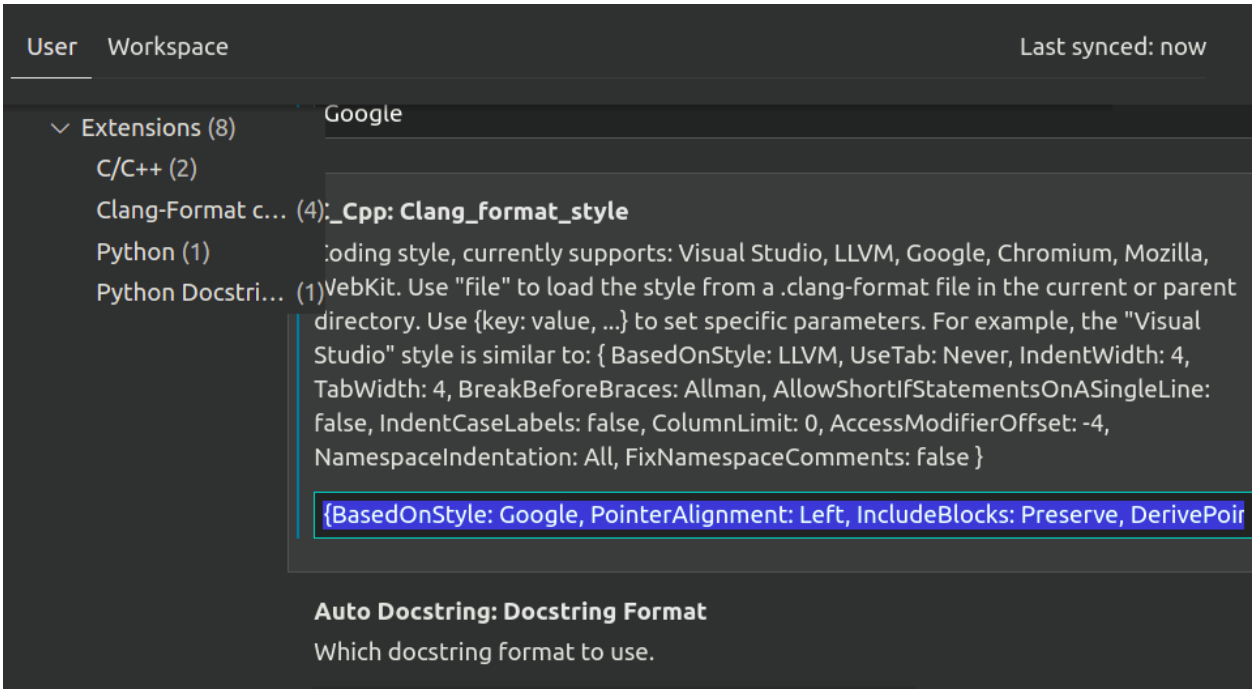
Coding convention for the code base should be formatted and checked by clang-format. Configuration file `.clang-format` for code formatting can be found here.

Setup for Visual Studio Code (VS Code)

VS Code can use clang-format to format source code file (Ctrl+Shift+i).

- Install Clang-Format extension in VS Code: https://marketplace.visualstudio.com/items?itemName=xaver.clang-format.
- Configure `C_Cpp:Clang_format_style` as following:

```
{BasedOnStyle: Google, PointerAlignment: Left, IncludeBlocks: Preserve,␣
↪DerivePointerAlignment: false}
```



Run code format for the whole project on Ubuntu (should be done before committing your code):

```
bash scripts/format_code.sh
```

17.1.2 Comments

- Use `//` to start a comment. This project use Doxygen to generate documentation automatically. Use `///` to start a comment that should be used to generate documentation.
- Add license to all `.h` and `.cpp` files like below example.

Example:

```
// Copyright 2021 The DaisyKit Authors.
//
// 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.

#ifndef DAISYKIT_GRAPHS_CORE_NODE_H_
#define DAISYKIT_GRAPHS_CORE_NODE_H_

#include "daisykit/common/types.h"
```

(continues on next page)

(continued from previous page)

```
#include "daisykit/graphs/core/connection.h"
#include "daisykit/graphs/core/node_type.h"
#include "daisykit/graphs/core/packet.h"

#include <atomic>
#include <chrono>
#include <map>
#include <memory>
#include <string>
#include <thread>

namespace daisykit {
namespace graphs {

/// A node is a processing unit which handle a task such as inferecing a model,
/// running an image processing operation, or visualizing data.
class Node {
 public:
  /// Node constructor. Passing a name and node type here.
  /// Synchronous nodes (kSyncNode) processing function Tick() is activated by
  /// the previous node, which means all processing pipeline runs node by node.
  /// Asynchronous node (kAsyncNode) has a processing thread inside to run
  /// processing Tick() in a loop. Thus, these node can run paralelly.
  Node(
      const std::string& node_name,  /// Node name
      NodeType node_type =
          NodeType::kAsyncNode  /// Node type / operation mode.
                                /// NodeType::kSyncNode for synchronous node,
                                /// NodeType::kAsyncNode for multithreading node
  );

  /// Activate a node. This function create and activate processing thread for
  /// asynchronous node.
  void Activate();

  /// Feed data to a node. This function can be used to feed data to the input
  /// node of a graph, where there is no connection in.
  void Input(const std::string& input_name, PacketPtr packet);

  /// Add an input connection to the node.
  /// Input connections are used to get input to this node.
  void AddInputConnection(std::shared_ptr<Connection> connection);

  ...

  /// Virtual method for processing data, needs to be implemented by derived
  /// classes. This method checks and gets all required data, processes data and
  /// outputs to out connections.
  virtual void Tick() = 0;

  /// Getters for node info
  std::string GetNodeName();
```

(continues on next page)

(continued from previous page)

```
  NodeType GetNodeType();

 protected:
  /// Prepare all needed input as a map for processing function.
  void PrepareInputs(std::map<std::string, PacketPtr>& input_map);

  /// Publish outputs to output connections.
  void Publish(const std::map<std::string, PacketPtr>& outputs);

 private:
  /// Worker thread for each node.
  /// This thread runs Tick() function in a loop
  void WorkerThread();

  /// Start processing thread.
  std::thread SpawnWorker();

  std::atomic<bool> is_activated_;
  std::thread worker_thread_;
  std::string node_name_;
  NodeType node_type_;
  std::vector<std::shared_ptr<Connection>> in_connections_;
  std::vector<std::shared_ptr<Connection>> out_connections_;
  std::thread processing_worker_;
};

}  // namespace graphs
}  // namespace daisykit

#endif
```

17.2 II. Contribution flow

- Create a separated branch for each task.
- Suggested naming for git branch:
  - For a new feature: `feature/example`
  - Code refactoring: `refactor/example`
  - Bugfix: `fix/example`
- The development flow should be:

```
Create a new branch for development => Write some code => Test your new code => Create a␣
↪pull request to master branch and add your teammates to review => Revise if needed =>␣
↪A teammate approves and merges your pull request after all reviewer approved.
```

17.3 III. Contribute to DaisyKit SDK

Create a pull request to https://github.com/DaisyLabSolutions/daisykit. Visit repository for the setup instructions.

Next tasks: Build model training code, inference code, design and build flow architecture, write documentation and tutorials.

17.4 IV. Contribute to DaisyKit Android

Create a pull request to https://github.com/DaisyLabSolutions/daisykit-android. Visit repository for the setup instructions.

Next tasks: Build wrappers for Kotlin, Java and example applications.

17.5 V. Contribute to Daisykit iOS

Create a pull request to https://github.com/DaisyLabSolutions/daisykit-ios. Visit repository for the setup instructions.

Next tasks: Build base app, wrappers for Swift, Objective-C and example applications.

CHAPTER

EIGHTEEN

INDICES AND TABLES

- genindex
- search