Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Домашнее задание №2
«Введение в CV на примере реализации задачи Key point detection на C++ и Python»
по курсу: «Языки и методы программирования»

Выполнил:
Студент группы ИУ9-21Б
Гречко Г.В.

Проверил:
Посевин Д.П.

Москва, 2022

# Содержание


- Содержание
- Цели
- Задачи
- Решение
  - Распознавание точек кисти
  - Распознование точек тела
  - Сравнение скорости работы алгоритма распознования кисти руки на **C++** и **python**
  - Сравнение скорости работы разных алгоритмов распознавания кисти руки на **python**
  - Изображения, которые использовались в ходе работы:
- Выводы


# Цели

Знакомство с возможностями языка C++ и Python для реализации задач машинного зрения.

# Задачи

Реализовать на C++ и Python под любую ОС по желанию студента следующие задачи:

1. Распознавание координат точек кисти со снимков получаемых с камеры, координаты точек выводятся списком в консоль в формате JSON.
2. Распознавание координат точек тела со снимков получаемых с камеры, координаты точек выводятся списком в консоль в формате JSON.
3. Сравнить скорость работы алгоритма распознавания кисти руки выполненного на C++ со скоростью распознавания выполненного на Python. В отчете привести сравнение скоростей.
4. Сравнить скорость распознавания кисти руки алгоритмом выполненным на языке Python в этом Модуле со скоростью алгоритма распознавания кисти руки на базе **Mediapipe** выполненным на языке Python в предыдущем **Модуле №1**. В отчете привести сравнение скоростей.
5. Сделать выводы.

# Решение

Мною был реализован вывод координат точек в JSON как для анализа снимков, так и для видео. Однако в отчете будет показан лишь результат вывода для анализа снимков, в том числе и с камеры.

## Распознавание точек кисти

Исходный код на **Python**:

**handPoseImage.py**

```
from __future__ import division
import cv2
import time
import numpy as np

protoFile = "hand/pose_deploy.prototxt"
weightsFile = "hand/pose_iter_102000.caffemodel"
nPoints = 22
```

```
POSE_PAIRS = [
    [0,1],[1,2],[2,3],[3,4],[0,5],[5,6],[6,7],[7,8],[0,9],[9,10],[10,11],[11,12],[
    ]
net = cv2.dnn.readNetFromCaffe(protoFile, weightsFile)

frame = cv2.imread("right-frontal.jpg")
frameCopy = np.copy(frame)
frameWidth = frame.shape[1]
frameHeight = frame.shape[0]
aspect_ratio = frameWidth/frameHeight

threshold = 0.1

t = time.time()
# input image dimensions for the network
inHeight = 368
inWidth = int(((aspect_ratio*inHeight)*8)//8)
inpBlob = cv2.dnn.blobFromImage(frame, 1.0 / 255, (inWidth, inHeight),
    (0, 0, 0), swapRB=False, crop=False)

net.setInput(inpBlob)

output = net.forward()
print("time taken by network : {:.3f}".format(time.time() - t))

# Empty list to store the detected keypoints
points = []
count = 0
for i in range(nPoints):
    # confidence map of corresponding body's part.
    probMap = output[0, i, :, :]
    probMap = cv2.resize(probMap, (frameWidth, frameHeight))

    # Find global maxima of the probMap.
    minVal, prob, minLoc, point = cv2.minMaxLoc(probMap)

    if prob > threshold :
        cv2.circle(frameCopy, (int(point[0]), int(point[1])), 8, (0, 255,
    255), thickness=-1, lineType=cv2.FILLED)
        cv2.putText(frameCopy, "{}".format(i), (int(point[0]),
    int(point[1])), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2,
    lineType=cv2.LINE_AA)

        # Add the point to the list if the probability is greater than
            the threshold
        points.append((int(point[0]), int(point[1])))
        count+=1
    else :
        points.append(None)

if count > 1:
    print ("Points: [")
    i = 0
    k = 0
    while k < count:
        if points[i] != None:
            print ("{\"id\":", k, "\", coords\":{", "\"x\":",
                points[i][0], "\"y\":", points[i][1], "}, ")
            k+=1
        i+=1
    #print ("{\"id\":", count - 1, "\", coords\":{", "\"x\":",
        points[count - 1][0], "\"y\":", points[count - 1][1], "} ")
```

```
    print ("]")
# Draw Skeleton
for pair in POSE_PAIRS:
    partA = pair[0]
    partB = pair[1]
    if points[partA] and points[partB]:
        cv2.line(frame, points[partA], points[partB], (0, 255, 255), 2)
        cv2.circle(frame, points[partA], 8, (0, 0, 255), thickness=-1,
    ↪ lineType=cv2.FILLED)
        cv2.circle(frame, points[partB], 8, (0, 0, 255), thickness=-1,
    ↪ lineType=cv2.FILLED)


cv2.imshow('Output-Keypoints', frameCopy)
cv2.imshow('Output-Skeleton', frame)


cv2.imwrite('Output-Keypoints.jpg', frameCopy)
cv2.imwrite('Output-Skeleton.jpg', frame)

print("Total time taken : {:.3f}".format(time.time() - t))

cv2.waitKey(0)
```

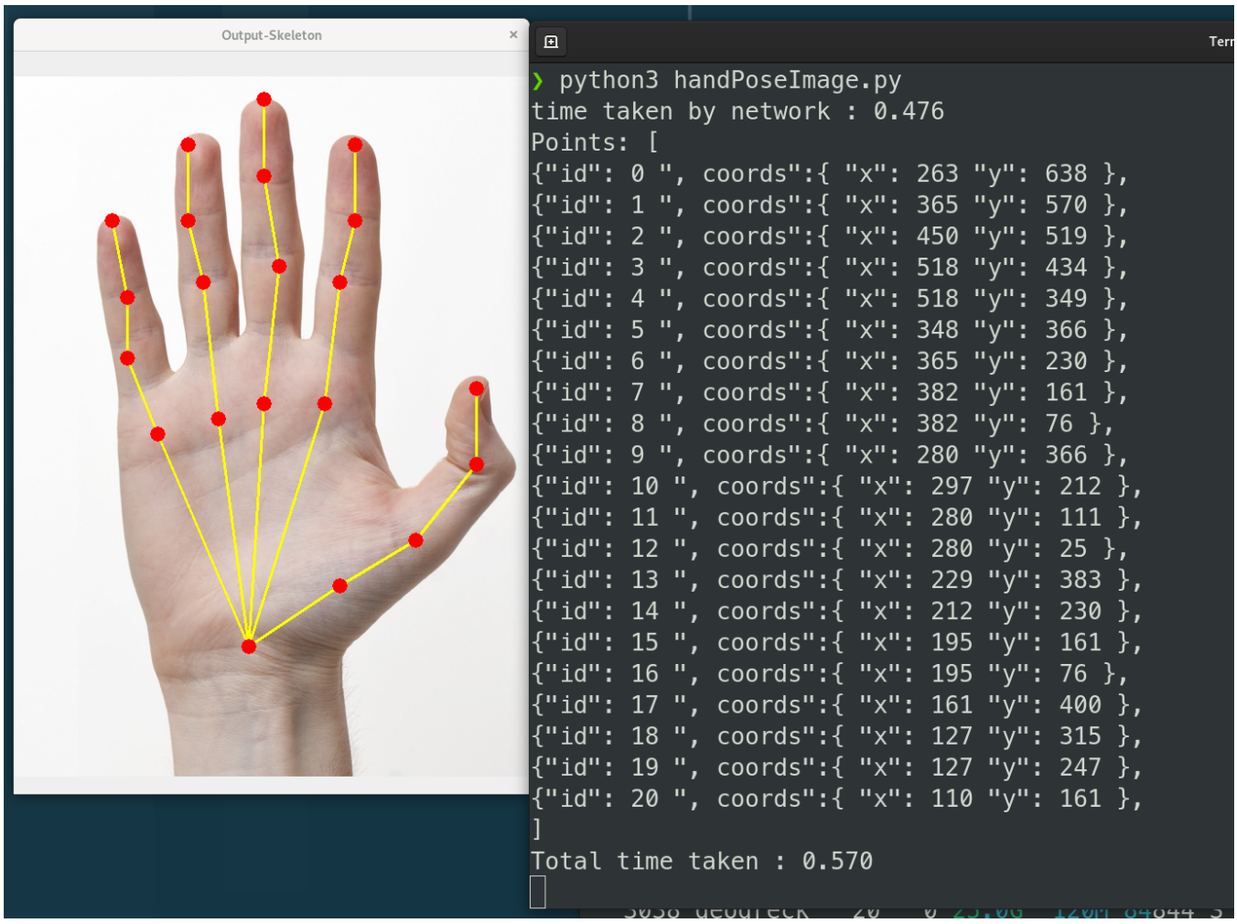


Рис. 1: Распознавание на Python

Исходный код на языке **C++**:

```
#include <opencv2/dnn.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <iostream>

using namespace std;
using namespace cv;
using namespace cv::dnn;
```

```
9
10
11  const int POSE_PAIRS[20][2] =
12  {
13      {0,1}, {1,2}, {2,3}, {3,4},         // thumb
14      {0,5}, {5,6}, {6,7}, {7,8},         // index
15      {0,9}, {9,10}, {10,11}, {11,12},    // middle
16      {0,13}, {13,14}, {14,15}, {15,16},  // ring
17      {0,17}, {17,18}, {18,19}, {19,20}   // small
18  };
19
20  string protoFile = "hand/pose_deploy.prototxt";
21  string weightsFile = "hand/pose_iter_102000.caffemodel";
22
23  int nPoints = 22;
24
25  int main(int argc, char **argv)
26  {
27
28      cout << "USAGE : ./handPoseImage <imageFile> " << endl;
29
30      string imageFile = "right-frontal.jpg";
31      // Take arguments from commmand line
32      if (argc == 2)
33      {
34        imageFile = argv[1];
35      }
36
37      float thresh = 0.01;
38
39      Mat frame = imread(imageFile);
40      Mat frameCopy = frame.clone();
41      int frameWidth = frame.cols;
42      int frameHeight = frame.rows;
43
44      float aspect_ratio = frameWidth/(float)frameHeight;
45      int inHeight = 368;
46      int inWidth = (int(aspect_ratio*inHeight) * 8) / 8;
47
48      cout << "inWidth = " << inWidth << " ; inHeight = " << inHeight <<
    ↪  endl;
49
50      double t = (double) cv::getTickCount();
51      Net net = readNetFromCaffe(protoFile, weightsFile);
52
53      Mat inpBlob = blobFromImage(frame, 1.0 / 255, Size(inWidth,
    ↪  inHeight), Scalar(0, 0, 0), false, false);
54
55      net.setInput(inpBlob);
56
57      Mat output = net.forward();
58
59      int H = output.size[2];
60      int W = output.size[3];
61
62      // find the position of the body parts
63      vector<Point> points(nPoints);
64      for (int n=0; n < nPoints; n++)
65      {
66          // Probability map of corresponding body's part.
67          Mat probMap(H, W, CV_32F, output.ptr(0,n));
68          resize(probMap, probMap, Size(frameWidth, frameHeight));
```

```
        Point maxLoc;
        double prob;
        minMaxLoc(probMap, 0, &prob, 0, &maxLoc);
        if (prob > thresh)
        {
            circle(frameCopy, cv::Point((int)maxLoc.x, (int)maxLoc.y), 8, Scalar(0,255,255), -1);
            cv::putText(frameCopy, cv::format("%d", n), cv::Point((int)maxLoc.x, (int)maxLoc.y), cv::FONT_HERSHEY_COMPLEX, 1, cv::Scalar(0, 0, 255), 2);

        }
        points[n] = maxLoc;
    }

    int nPairs = sizeof(POSE_PAIRS)/sizeof(POSE_PAIRS[0]);

    if (nPairs > 0){
        cout<<"Points: [";
    }
    for (int n = 0; n < nPairs; n++)
    {
        // lookup 2 connected body/hand parts
        Point2f partA = points[POSE_PAIRS[n][0]];
        Point2f partB = points[POSE_PAIRS[n][1]];

        if (partA.x<=0 || partA.y<=0 || partB.x<=0 || partB.y<=0)
            continue;

        cout<<"{\"id\":"<< n << "\", coords\":{"<< "\"x\":"<< points[n].x<< "\"y\":"<< points[n].y<< "}, ";

        line(frame, partA, partB, Scalar(0,255,255), 8);
        circle(frame, partA, 8, Scalar(0,0,255), -1);
        circle(frame, partB, 8, Scalar(0,0,255), -1);
    }
    if (nPairs > 0){
        cout<<"]\n";
    }

    t = ((double)cv::getTickCount() - t)/cv::getTickFrequency();
    cout << "Time Taken = " << t << endl;
    imshow("Output-Keypoints", frameCopy);
    imshow("Output-Skeleton", frame);
    imwrite("Output-Skeleton.jpg", frame);

    waitKey();

    return 0;
}
```

## Распознование точек тела

Исходный код на **Python**:

```
import cv2
import time
import numpy as np
import argparse

parser = argparse.ArgumentParser(description='Run keypoint detection')
```

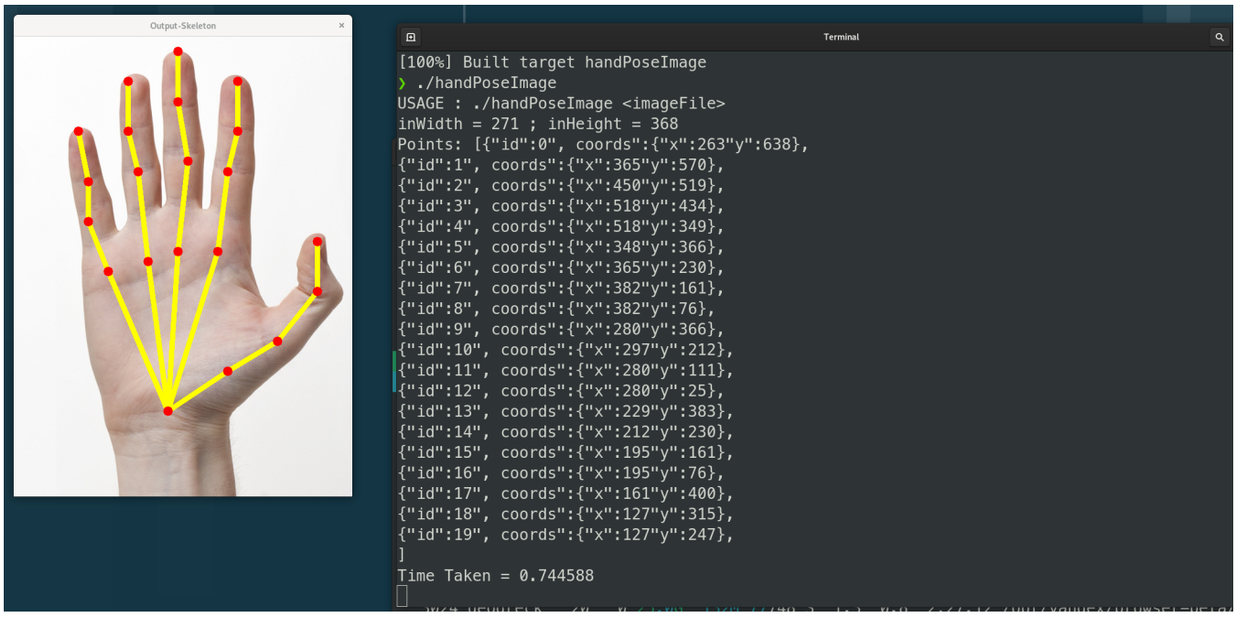


Рис. 2: Распознавание на C++

```
parser.add_argument("--device", default="cpu", help="Device to inference on")
parser.add_argument("--image_file", default="single.jpeg", help="Input image")

args = parser.parse_args()


MODE = "COCO"

if MODE is "COCO":
    protoFile = "pose/coco/pose_deploy_linevec.prototxt"
    weightsFile = "pose/coco/pose_iter_440000.caffemodel"
    nPoints = 18
    POSE_PAIRS = [
 [1,0],[1,2],[1,5],[2,3],[3,4],[5,6],[6,7],[1,8],[8,9],[9,10],[1,11],[11,12],[1

elif MODE is "MPI" :
    protoFile = "pose/mpi/pose_deploy_linevec_faster_4_stages.prototxt"
    weightsFile = "pose/mpi/pose_iter_160000.caffemodel"
    nPoints = 15
    POSE_PAIRS = [[0,1], [1,2], [2,3], [3,4], [1,5], [5,6], [6,7],
 [1,14], [14,8], [8,9], [9,10], [14,11], [11,12], [12,13] ]


frame = cv2.imread(args.image_file)
frameCopy = np.copy(frame)
frameWidth = frame.shape[1]
frameHeight = frame.shape[0]
threshold = 0.1

net = cv2.dnn.readNetFromCaffe(protoFile, weightsFile)

if args.device == "cpu":
    net.setPreferableBackend(cv2.dnn.DNN_TARGET_CPU)
    print("Using CPU device")
elif args.device == "gpu":
    net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
    net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)
    print("Using GPU device")

t = time.time()
```

```
# input image dimensions for the network
inWidth = 368
inHeight = 368
inpBlob = cv2.dnn.blobFromImage(frame, 1.0 / 255, (inWidth, inHeight),
                          (0, 0, 0), swapRB=False, crop=False)

net.setInput(inpBlob)

output = net.forward()
print("time taken by network : {:.3f}".format(time.time() - t))

H = output.shape[2]
W = output.shape[3]

# Empty list to store the detected keypoints
points = []
count = 0

for i in range(nPoints):
    # confidence map of corresponding body's part.
    probMap = output[0, i, :, :]

    # Find global maxima of the probMap.
    minVal, prob, minLoc, point = cv2.minMaxLoc(probMap)

    # Scale the point to fit on the original image
    x = (frameWidth * point[0]) / W
    y = (frameHeight * point[1]) / H

    if prob > threshold :
        cv2.circle(frameCopy, (int(x), int(y)), 8, (0, 255, 255),
    ↪ thickness=-1, lineType=cv2.FILLED)
        cv2.putText(frameCopy, "{}".format(i), (int(x), int(y)),
    ↪ cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, lineType=cv2.LINE_AA)

        # Add the point to the list if the probability is greater than
         ↪  the threshold
        points.append((int(x), int(y)))
        count+=1
    else :
        points.append(None)

# JSON output
if count > 1:
    print ("Points: [")
    i = 0
    k = 0
    while k < count:
        if points[i] != None:
            print ("{\"id\":", k, "\", coords\":{", "\"x\":",
             ↪  points[i][0], "\"y\":", points[i][1], "}, ")
            k+=1
        i+=1
    #print ("{\"id\":", count - 1, "\", coords\":{", "\"x\":",
     ↪  points[count - 1][0], "\"y\":", points[count - 1][1], "} ")
    print ("]")

# Draw Skeleton
for pair in POSE_PAIRS:
    partA = pair[0]
    partB = pair[1]

```

```
    if points[partA] and points[partB]:
        cv2.line(frame, points[partA], points[partB], (0, 255, 255), 2)
        cv2.circle(frame, points[partA], 8, (0, 0, 255), thickness=-1,
    ↪ lineType=cv2.FILLED)


cv2.imshow('Output-Keypoints', frameCopy)
cv2.imshow('Output-Skeleton', frame)


cv2.imwrite('Output-Keypoints.jpg', frameCopy)
cv2.imwrite('Output-Skeleton.jpg', frame)

print("Total time taken : {:.3f}".format(time.time() - t))

cv2.waitKey(0)
```

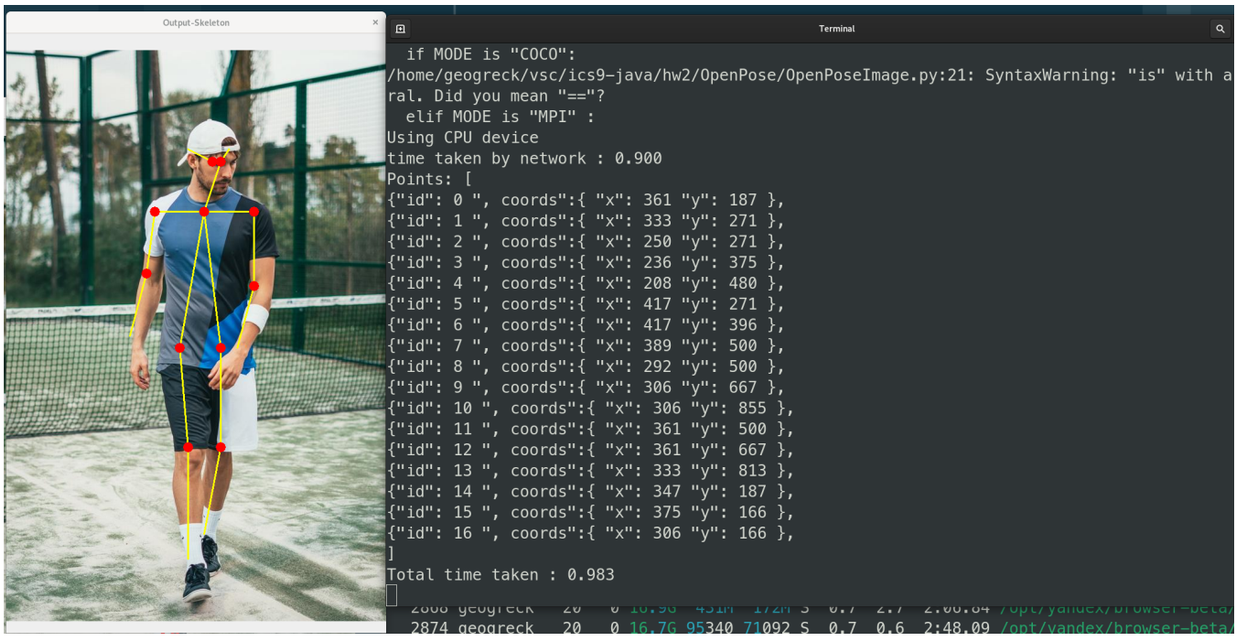


Рис. 3: Распознование на Python

Исходный код на **C++**:

```
#include <opencv2/dnn.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <iostream>

using namespace std;
using namespace cv;
using namespace cv::dnn;

#define MPI

#ifdef MPI
const int POSE_PAIRS[14][2] =
{
    {0,1}, {1,2}, {2,3},
    {3,4}, {1,5}, {5,6},
    {6,7}, {1,14}, {14,8}, {8,9},
    {9,10}, {14,11}, {11,12}, {12,13}
};

string protoFile =
    ↪ "pose/mpi/pose_deploy_linevec_faster_4_stages.prototxt";
string weightsFile = "pose/mpi/pose_iter_160000.caffemodel";
```

```
23
24  int nPoints = 15;
25  #endif
26
27  #ifdef COCO
28  const int POSE_PAIRS[17][2] =
29  {
30      {1,2}, {1,5}, {2,3},
31      {3,4}, {5,6}, {6,7},
32      {1,8}, {8,9}, {9,10},
33      {1,11}, {11,12}, {12,13},
34      {1,0}, {0,14},
35      {14,16}, {0,15}, {15,17}
36  };
37
38  string protoFile = "pose/coco/pose_deploy_linevec.prototxt";
39  string weightsFile = "pose/coco/pose_iter_440000.caffemodel";
40
41  int nPoints = 18;
42  #endif
43
44  int main(int argc, char **argv)
45  {
46
47      cout << "USAGE : ./OpenPose <imageFile> " << endl;
48      cout << "USAGE : ./OpenPose <imageFile> <device>" << endl;
49
50      string device = "cpu";
51
52      string imageFile = "single.jpeg";
53      // Take arguments from commmand line
54      if (argc == 2)
55      {
56        if((string)argv[1] == "gpu")
57          device = "gpu";
58        else
59        imageFile = argv[1];
60      }
61      else if (argc == 3)
62      {
63          imageFile = argv[1];
64          if((string)argv[2] == "gpu")
65              device = "gpu";
66      }
67
68
69
70      int inWidth = 368;
71      int inHeight = 368;
72      float thresh = 0.1;
73
74      Mat frame = imread(imageFile);
75      Mat frameCopy = frame.clone();
76      int frameWidth = frame.cols;
77      int frameHeight = frame.rows;
78
79      double t = (double) cv::getTickCount();
80      Net net = readNetFromCaffe(protoFile, weightsFile);
81
82      if (device == "cpu")
83      {
84          cout << "Using CPU device" << endl;
```

```
85          net.setPreferableBackend(DNN_TARGET_CPU);
86      }
87      else if (device == "gpu")
88      {
89          cout << "Using GPU device" << endl;
90          net.setPreferableBackend(DNN_BACKEND_CUDA);
91          net.setPreferableTarget(DNN_TARGET_CUDA);
92      }
93
94      Mat inpBlob = blobFromImage(frame, 1.0 / 255, Size(inWidth,
    ↪ inHeight), Scalar(0, 0, 0), false, false);
95
96      net.setInput(inpBlob);
97
98      Mat output = net.forward();
99
100     int H = output.size[2];
101     int W = output.size[3];
102
103     // find the position of the body parts
104     vector<Point> points(nPoints);
105     for (int n=0; n < nPoints; n++)
106     {
107         // Probability map of corresponding body's part.
108         Mat probMap(H, W, CV_32F, output.ptr(0,n));
109
110         Point2f p(-1,-1);
111         Point maxLoc;
112         double prob;
113         minMaxLoc(probMap, 0, &prob, 0, &maxLoc);
114         if (prob > thresh)
115         {
116             p = maxLoc;
117             p.x *= (float)frameWidth / W ;
118             p.y *= (float)frameHeight / H ;
119
120             circle(frameCopy, cv::Point((int)p.x, (int)p.y), 8,
    ↪ Scalar(0,255,255), -1);
121             cv::putText(frameCopy, cv::format("%d", n),
    ↪ cv::Point((int)p.x, (int)p.y), cv::FONT_HERSHEY_COMPLEX, 1,
    ↪ cv::Scalar(0, 0, 255), 2);
122
123         }
124         points[n] = p;
125     }
126
127     int nPairs = sizeof(POSE_PAIRS)/sizeof(POSE_PAIRS[0]);
128
129     if (nPairs > 0){
130         cout<<"Points: [";
131     }
132     for (int n = 0; n < nPairs; n++)
133     {
134         // lookup 2 connected body/hand parts
135         Point2f partA = points[POSE_PAIRS[n][0]];
136         Point2f partB = points[POSE_PAIRS[n][1]];
137
138         if (partA.x<=0 || partA.y<=0 || partB.x<=0 || partB.y<=0)
139             continue;
140
141         cout<<"{\"id\":"<< n << "\", coords\":{"<< "\"x\":"<<
    ↪ points[n].x<< "\"y\":"<< points[n].y<< "}, ";
```

```
142
143         line(frame, partA, partB, Scalar(0,255,255), 8);
144         circle(frame, partA, 8, Scalar(0,0,255), -1);
145         circle(frame, partB, 8, Scalar(0,0,255), -1);
146     }
147     if (nPairs > 0){
148         cout<<"]\n";
149     }
150
151     t = ((double)cv::getTickCount() - t)/cv::getTickFrequency();
152     cout << "Time Taken = " << t << endl;
153     imshow("Output-Keypoints", frameCopy);
154     imshow("Output-Skeleton", frame);
155     imwrite("Output-Skeleton.jpg", frame);
156
157     waitKey();
158
159     return 0;
160 }
```

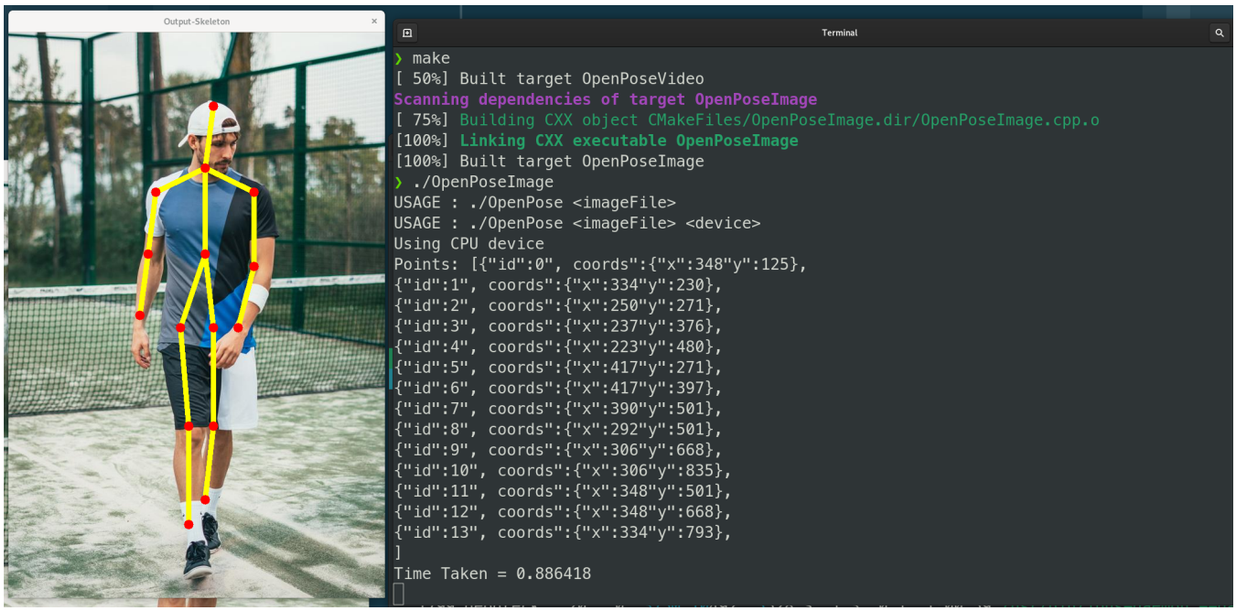


Рис. 4: Реализация на C++

## Сравнение скорости работы алгоритма распознования кисти руки на C++ и python

Для измерения использовалась команда linux **time**. В таблицу заносилось общее(total) время.

| | picture1 | picture2 | picture3 |
|---|---|---|---|
| python | 0.750 | 1.401 | 1.633 |
| c++ | 0.713 | 1.411 | 1.643 |

Программы работают за примерно одинаковое время.

## Сравнение скорости работы разных алгоритмов распознавания кисти руки на `python`

Методика тестирования та же.

| | picture1 | picture2 | picture3 |
|---|---|---|---|
| 1 module | 0.785 | 0.769 | 0.734 |
| 2 module | 0.846 | 1.463 | 1.652 |

Первая программа на более сложных изображениях работает ощутимо быстрее.

## Изображения, которые использовались в ходе работы:

Рис. 5: picture 1

Рис. 6: picture 2

Рис. 7: picture 3

## Выводы

Существенной разницы в скорости работы распознования с помощью нейронной сети на **C++** и **python** нет.

Однако нейронная сеть работает гораздо медленее алгоритма распознования изображения из 1 модуля.

Но у нее есть свои преимущества, например, нейронная сеть дает более точный результат, который можно ещё улучшить путем ее дальнейшего обучения.

Алгоритм первого модуля в свою очередь позволяет обрабатывать изображение в реальном времени, благодаря своей скорости.