讨论内容

- 已加入GITHUB repo。
- 服务器都能正常使用。

个人工作

MINGRUIBO

- 服务器使用说明。
- 数据集 `REDS` 说明。
- 阅读 `EDVR` 论文并分享。
  - https://github.com/xinntao/EDVR
  - EDVR = PCD(Pyramid, Cascading and Deformable alignment module) + TSA(Temporal and Spatial Attention fusion module)
  - trick: 2-Stage Restoration

ZHENGYUNLONG

- 分享光流法

  光流基本知识

  - 光流（optical flow）是空间运动物体在观察成像平面上的像素运动的瞬时速度。
  - 光流法是利用图像序列中像素在时间域上的变化以及相邻帧之间的相关性来找到上一帧跟当前帧之间存在的对应关系，从而计算出相邻帧之间物体的运动信息的一种方法。
  - 光流矢量，二维图像平面特定坐标点上的灰度瞬时变化率，包括x,y方向。
  - 光流场，光流矢量的集合，可视为带有灰度的像素点在图像平面上运动而产生的瞬时速度场。理想情况下，光流场对应运动场。

  光流基本算法

  - 常见做法有基于梯度的方法、基于匹配的方法、基于能量的方法、基于相位的方法、神经动力学方法。

  稠密光流与稀疏光流

  - 稠密光流是针对整个图像或指定的某一片区域进行**逐点**匹配的图像配准方法，它计算图像上所有的点的偏移量，从而形成一个稠密的光流场。通过这个稠密的光流场，可以进行像素级别的图像配准。
  - 稀疏光流则是指定一组特征点进行跟踪，显然稀疏光流计算开销要小得多，当然其配准效果没有稠密光流好。

  代码实现示例

  - 这里贴出使用OpenCV光流场方法标出前后景（运动与静止）的关键代码：

```
#include <iostream>
#include "opencv2/opencv.hpp"
#include <opencv2\imgproc\types_c.h>

using namespace cv;
using namespace std;
```

```
#define UNKNOWN_FLOW_THRESH 1e9
void makecolorwheel(vector<Scalar>& colorwheel)
{
    int RY = 15;
    int YG = 6;
    int GC = 4;
    int CB = 11;
    int BM = 13;
    int MR = 6;

    int i;

    for (i = 0; i < RY; i++) colorwheel.push_back(Scalar(255, 255 * i / RY,
0));
    for (i = 0; i < YG; i++) colorwheel.push_back(Scalar(255 - 255 * i / YG,
255, 0));
    for (i = 0; i < GC; i++) colorwheel.push_back(Scalar(0, 255, 255 * i /
GC));
    for (i = 0; i < CB; i++) colorwheel.push_back(Scalar(0, 255 - 255 * i /
CB, 255));
    for (i = 0; i < BM; i++) colorwheel.push_back(Scalar(255 * i / BM, 0,
255));
    for (i = 0; i < MR; i++) colorwheel.push_back(Scalar(255, 0, 255 - 255 *
i / MR));
}

void motionToColor(Mat flow, Mat& color)
{
    if (color.empty())
        color.create(flow.rows, flow.cols, CV_8UC3);

    static vector<Scalar> colorwheel; //Scalar r,g,b
    if (colorwheel.empty())
        makecolorwheel(colorwheel);

    // determine motion range:
    float maxrad = -1;

    // Find max flow to normalize fx and fy
    for (int i = 0; i < flow.rows; ++i)
    {
        for (int j = 0; j < flow.cols; ++j)
        {
            Vec2f flow_at_point = flow.at<Vec2f>(i, j);
            float fx = flow_at_point[0];
            float fy = flow_at_point[1];
            if ((fabs(fx) > UNKNOWN_FLOW_THRESH) || (fabs(fy) >
UNKNOWN_FLOW_THRESH))
                continue;
            float rad = sqrt(fx * fx + fy * fy);
            maxrad = maxrad > rad ? maxrad : rad;
        }
    }

    for (int i = 0; i < flow.rows; ++i)
    {
        for (int j = 0; j < flow.cols; ++j)
```

```
        {
            uchar* data = color.data + color.step[0] * i + color.step[1] *
j;
            Vec2f flow_at_point = flow.at<Vec2f>(i, j);

            float fx = flow_at_point[0] / maxrad;
            float fy = flow_at_point[1] / maxrad;
            if ((fabs(fx) > UNKNOWN_FLOW_THRESH) || (fabs(fy) >
UNKNOWN_FLOW_THRESH))
            {
                data[0] = data[1] = data[2] = 0;
                continue;
            }
            float rad = sqrt(fx * fx + fy * fy);

            float angle = atan2(-fy, -fx) / CV_PI;
            float fk = (angle + 1.0) / 2.0 * (colorwheel.size() - 1);
            int k0 = (int)fk;
            int k1 = (k0 + 1) % colorwheel.size();
            float f = fk - k0;
            //f = 0; // uncomment to see original color wheel

            for (int b = 0; b < 3; b++)
            {
                float col0 = colorwheel[k0][b] / 255.0;
                float col1 = colorwheel[k1][b] / 255.0;
                float col = (1 - f) * col0 + f * col1;
                if (rad <= 1)
                    col = 1 - rad * (1 - col); // increase saturation with
radius
                else
                    col *= .75; // out of range
                data[2 - b] = (int)(255.0 * col);
            }
        }
    }
}

int main(int, char**)
{
    VideoCapture cap;
    //cap.open(0);
    cap.open("srcVideo.mp4");

    if (!cap.isOpened())
        return -1;

    Mat prevgray, gray, flow, cflow, frame;

    Mat motion2color;

    for (;;)
    {
        double t = (double)cv::getTickCount();

        cap >> frame;
        cvtColor(frame, gray, CV_BGR2GRAY);
        imshow("src 1210", frame);
```

```
        if (prevgray.data)
        {
            calcOpticalFlowFarneback(prevgray, gray, flow, 0.5, 3, 15, 3, 5,
1.2, 0);//key code
            motionToColor(flow, motion2color);//key code
            imshow("dst 1210", motion2color);
        }
        if (waitKey(10) >= 0)
            break;
        std::swap(prevgray, gray);

        t = (double)cv::getTickCount() - t;
        cout << "cost time: " << t / ((double)cv::getTickFrequency() *
1000.) << endl;
    }
    return 0;
}
```

- 注意
  - 其中main函数部分标注的两行代码为关键代码，其代码意义及参数的使用需要进一步学习。
  - 会上还讨论了光流场的可视化问题，但我们视频超分应该不涉及可视化问题，只需记录光流场信息（即所标注第一行代码），因此这里略去。

参考

- https://blog.csdn.net/qq_41368247/article/details/82562165
- https://blog.csdn.net/a83025273/article/details/101955800