

一、实验一 dct 方式与效果

1. 实验设计

(1) 预处理：彩色转灰度

OpenCV 库支持直接将图片读取成灰度图，也可以将彩色图转换为灰度图。

此处直接将图片读取为灰度图。

(2) DCT 变换

这里实现了三种 DCT 方式，其中一种是 opencv 自带的 2d-dct 操作，一种是用 1d-dct 对原图先按列再按行做变换，一种是将原图分成 8×8 的块后每一块分别用 2d-dct 变换。

(3) 系数筛选与处理

在 DCT 后可能要对系数矩阵进行各类处理，主要分为以下几种：

- 不做处理
- 低通滤波
- 按 8×8 块低通滤波
- 量化、反量化

(4) IDCT 变换

分别对应三种 DCT 方式实现了与其对应的 IDCT 过程。

2. 性能对比

以下误差均基于 float32 类型运算。

	MSE	PSNR	用时(ms)
1d-dct by R & C	3.207169e-10	143.0695851883563	32.9056
2d-dct	3.207169e-10	143.0695851883563	3.9897
2d-dct by 8×8 block	1.5986934e-12	166.0931517886387	35.9387







注意到在相同精度的情况下，用 1d-DCT 时先列再行和先行再列得到的结果是相同的，和 2d-dct 的结果也是相同的，因为这两种算法本质上是等价的。两者数学意义上的等价性并非本次实验重点，不做展开。

而精度上将图像切分成 8×8 的块后再做 dct 要比直接做全图 dct 的准确性更高，这也容易理解， 8×8 的块进行 dct 后的系数矩阵可以更精准地拟合这小范围内的像素，只需要整合局部的信息。

3. 反思

由于低频信息比高频信息有更大的幅度，也通常携带了更多的信息，当只用少量的系数表达原图时我们就优先采用低频的系数，即矩阵左上角的元素。以下分别对全图 2d-dct(1d-dct 是一样的，故略去)和 8×8 块上的 2d-dct 分别采用低通滤波：

系数	2d-dct	2d-dct on 8×8 block
----	--------	------------------------------

1/4		
1/16		
1/64		

注意到全图 2d-dct 在系数减少时，其失真体现为模糊和“衍射”，而 8×8 块 2d-dct 在系数减少时，其失真体现为“马赛克”。

在全图 2d-dct 中，失去大量高频信息会导致一些边缘特征和局部特征丧失，因为图像中的灰度将不再有由高频信号引起的剧烈变化；所以图像呈现边缘不清晰。同时低频信号幅度较高，本身就会产生一些较宽的“波纹”，没有高频信号中和之后这些波纹便体现了出来。

而在 8×8 块 2d-dct 中，每一个 8×8 块中的信息随着系数比例降低大量流失，在 $1/64$ 的系数比时每个块中只剩下了一个信息也就是这个块的平均灰度，每个块只剩下一个纯色色块，所以相当于是将原图的分辨率降低到原来的 $1/64$ 又放大了 64 倍。

二、实验二 量化重要性

1. 实验设计

本实验和实验一采用相同的代码框架，不同的是在系数处理一步中不是用截取而是用量化进行处理。

一共设计了四种量化表，一种是 Jpeg Standard 量化表，一种是 Canon IXUS 60 量化表，一种是 Nikon CoolPix L12 量化表，以及一种自己生成的很粗糙的测试用量化表。

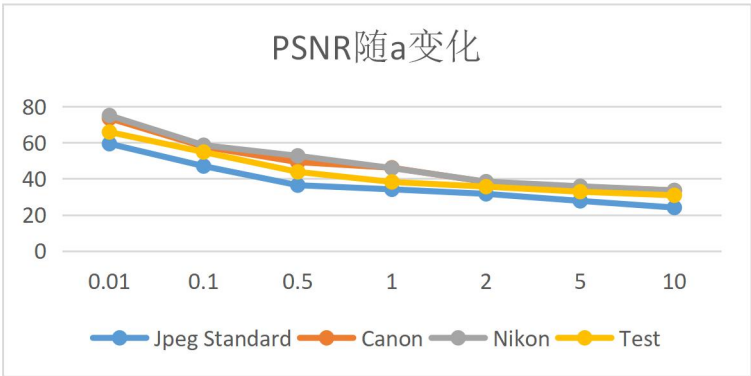
程序通过读取运行参数可以给量化表先乘上一个常数再进行量化，并输出其量化与反量化后的损失。

2. 反思

(1) 量化表系数

以下对四种量化表分别以不同的系数进行量化，并记录其 PSNR。

PSNR	0.01	0.1	0.5	1	2	5	10
Jpeg Standard	59.4	47.0	36.4	34.1	31.6	27.7	24.0
Canon	73.5	57.8	49.2	46.1	38.2	35.5	33.5
Nikon	75.1	58.6	52.7	45.9	38.4	35.8	33.5
Test	65.9	54.8	43.8	38.2	35.6	32.8	30.8



	A=0.01	A=1	A=10
图像			

可以观察到随着 a 越来越大，图像失真越来越严重，PSNR 越来越小；每一个块更趋向于平滑变化（高频信息丢失越来越多）。

(2) 影响量化表的因素及改进思路

观察到量化表中左上角的元素较小、右下的元素较大，而 block 经过 dct 后的值矩阵中左上角的元素较大右下的元素较小。可以想到量化表的意义便在于尽量保留矩阵左上角的元素并将右下的元素抹成 0。这样既方便编码也容易保真。

基于这种思想，一种可行的改进思路便是收集较多的图片，统计一个量化表在这个数据集上的表现，并横向比较各个量化表的性能（保真和平均编码长度），选取性能较好的量化表。

三、实验三 运动估计的直观体现

1. 实验设计

(1) 视频读写

Opencv 库中提供了视频读写的接口（读：VideoCapture，写：VideoWriter），可以逐帧读取视频中的帧。

为了直观呈现实验结果，即匹配块是否与直观认知相符，将匹配块边界在视频中标出。



(2) 选中目标块

手动在第 20 帧中标出白色车的车头（其在视频中持续存在时间较长）。将该部分灰度值子图从原帧中截出来作为“模式图”。

(3) 寻找匹配块

最初使用全搜索匹配，速度较慢；后设置了一个大小为 30 像素的可行域框，第 $i+1$ 帧的匹配块在第 i 块匹配块的外延 30 像素区域中查找。

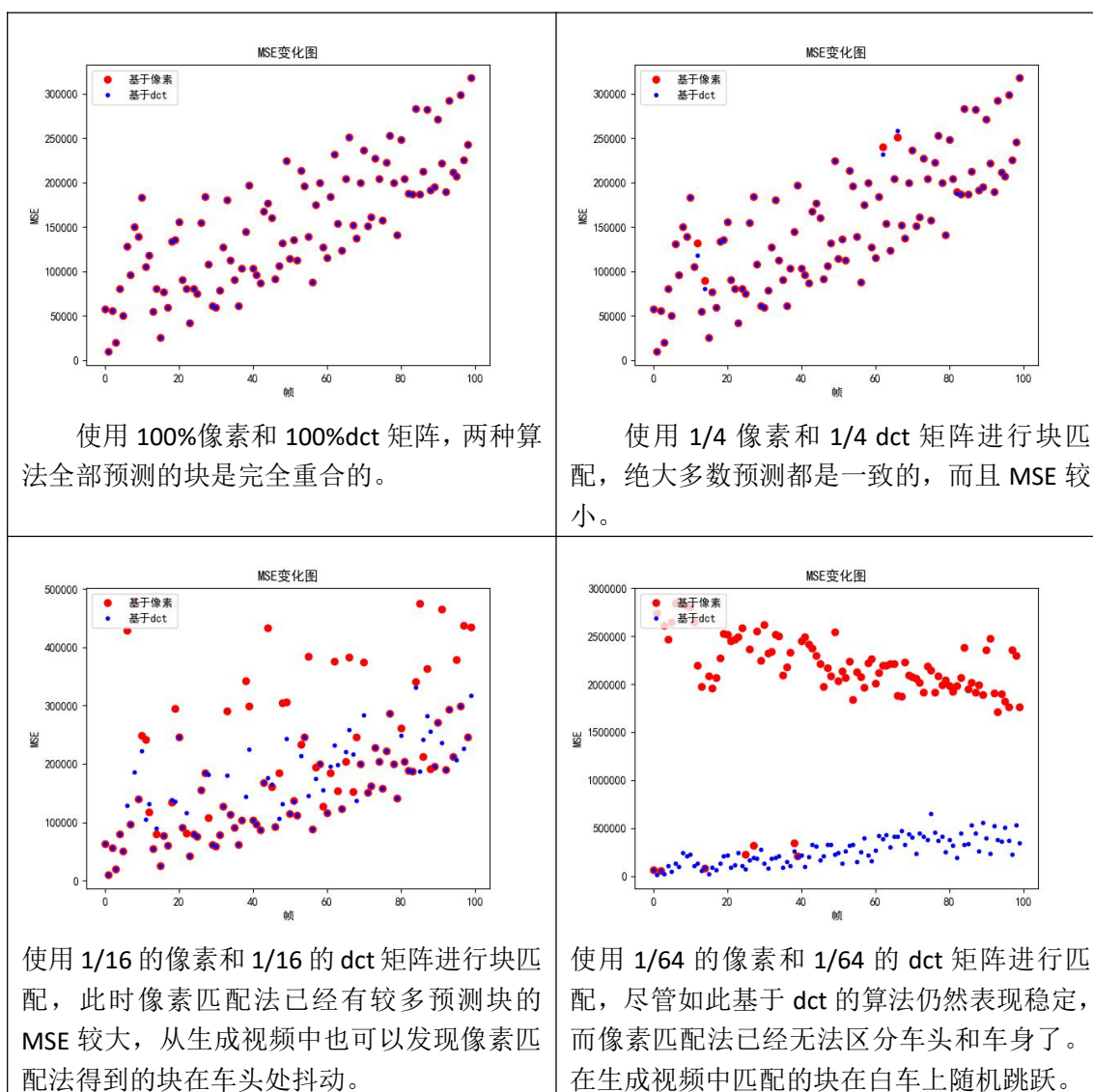
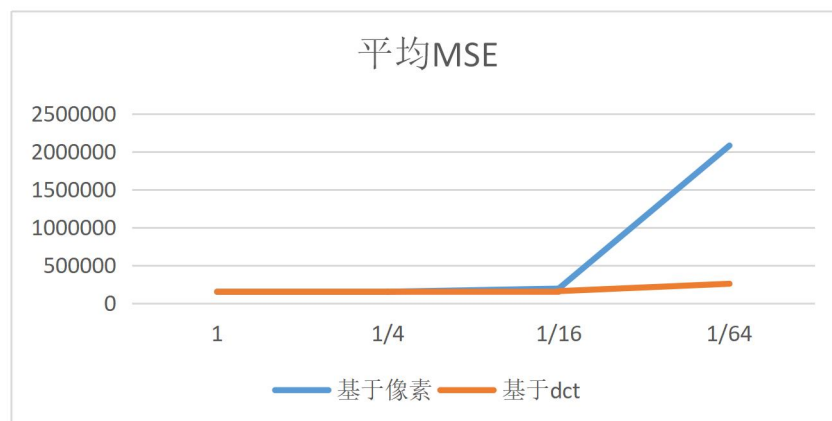
一开始在像素匹配时会出现错误匹配的问题，观察发现目标子图与模式子图的元素竟然都是 uint8 类型，当减出负数例如 -1 时自动模 256 取余成了正数 255，导致本来应该很小的平方和变成了很大的数。将子图的数据类型统一成 int32 后问题解决。

- 基于像素的块匹配
计算两个块的差矩阵，求其各元素平方和，选取该平方和最小的块作为匹配块。
- 基于 dct 的块匹配
计算块的 dct 矩阵，用 dct 矩阵之差计算各元素平方和，选取平方和最小的块作为匹配块。

2. 性能对比与反思

在上述选中块的前提下，两种算法的 MSE 的均值随信息比例变化以及 MSE 随帧变化图如下。

像素百分比	100%	1/4	1/16	1/64
基于像素平均 MSE	151591.91	151983.05	193185.5	2080878.31
基于 dct 平均 MSE	151591.91	151706.26	159840.33	257210.07



除了像素本身或频域信息外，该块和周围块在上一帧中的移动向量可以作为参考；也可以引入图像识别的语义信息辅助预测。

四、代码结构说明

代码仓库链接: <https://github.com/thomount/IED-VED.git>

```
-- src
--   ex1
--     Main.py          实验一、二流程
--     Func.py          功能函数实现
--   ex2
--     <空>              在 ex1 中一并实现
--   ex3
--     Main.py          实验三流程
--     Func.py          功能函数实现
--     Process.py       后处理生成统计图像
--     Init.py          初始化, 建立路径
-- output
--   ex1                生成图片和统计数据
--   ...
--   ex2
--   ex3                生成的标注视频和统计数据
--   ...
-- data
--   Lena.bmp
--   Cars.avi
-- Readme.md            实验中任务列表和日志
-- Requirements.txt     依赖库, 用 pip install -r requirements.txt 安装
-- Report.pdf           你猜
```