

Государственное образовательное учреждение высшего профессионального образования
Санкт-Петербургский национальный исследовательский университет
Информационных Технологий, Механики и Оптики
Факультет инфокоммуникационных технологий

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1
По Мультимедиа Технологиям
На тему: «Сжатие изображений»

Проверил(а): Хлопотов М.В.

Дата: «__»____201__г.

Оценка: _____

Работу выполнил(а):

Никончук А.П.
Студент(ка) группы К3242
Дневного отделения

Цель: Изучить алгоритмы стажития изображения

Задачи:

1. Выполнить реализацию алгоритмов квантования, децимации изображений;
2. Оценить результаты работы алгоритмов.

Выполнение работы:

При конвертировании изображений в разные форматы, конкретнее для уменьшения объема занимаемого изображением пространства происходит сжатие изображений. При этом применяются различные методы решений вопроса. Один из них – уменьшение цветовой палитры, решаемая с использованием процедуры квантования. Цель квантования состоит в отображении числового сигнала с областью значений X в квантованный сигнал области Y с уменьшенным числом значений. Это дает возможность представить квантованные величины с меньшим числом бит по сравнению с исходным.

Задача 1:

В цифровом пространстве greyscale требуется реализовать алгоритм скалярного равномерного квантования с переменной скоростью для постепенного уменьшения уровней яркости изображения. Сохранить изображение, в котором количество яркости многоменее 256, при этом с визуально приемлемым качеством.

Равномерное квантование – разбиение диапазона значений отсчетов сигнала на отрезки равной длины и замена этих значений на ближайший уровень квантования. То есть в при процедуре квантования происходит распределение градаций яркости на меньшее количество элементов шкалы, равномерное распределение при этом создает контрастность, также при условии взаимно-однозначного соответствия исходного и выходящего изображений, где скорость – это изменяемый шаг квантования. При известном количестве желаемых уровней квантования формула равномерного (однородного) квантования:

$$FQ = \text{round}\left(\frac{X}{QP}\right), \quad Y = FQ \cdot QP \quad (1)$$

где QP – шаг квантования. Тес уровни квантованных выходов расположены на одинаковых расстояниях QP друг от друга.

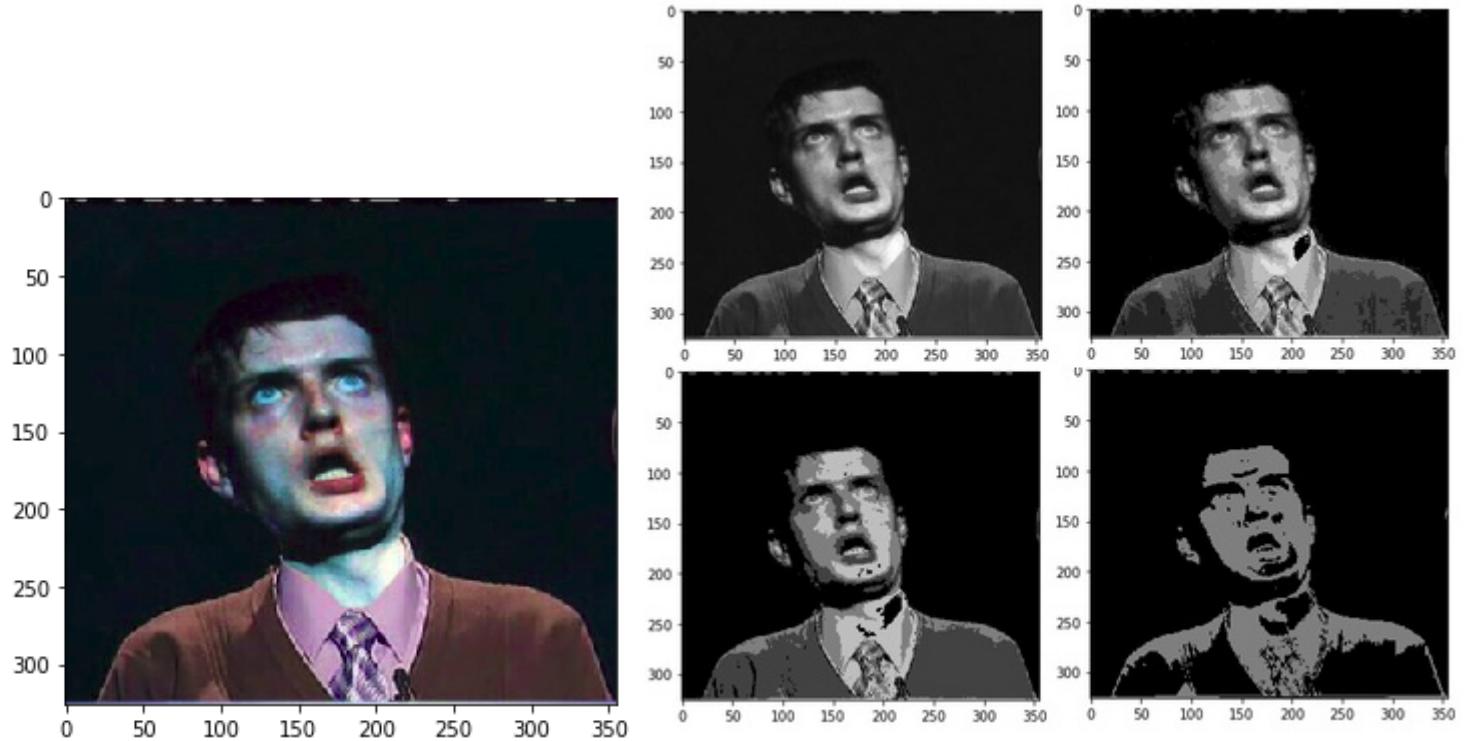
Функция прореживания (расчесывания) градаций яркости изображения:

Рисунок 1. Скриншот императивом функции прореживания и её использования

```
In [183]: def bright_lower(img, QP):
    #x = img_as_float(img)
    x = img.astype('float64')
    FQ = round(x/QP)
    y = FQ * QP
    #y = clip(y, 0, 1)
    y = y.astype('uint8')
    y = clip(y, 0, 255)
    #y = img_as_ubyte(y)
    return y
```

```
In [147]: img3 = imread('hw.jpg')
imshow(img3)
img3 = img_as_ubyte(color.rgb2gray(img3))
b8 = bright_lower(img3, 256)
b7 = bright_lower(img3, 128)
b6 = bright_lower(img3, 64)
b5 = bright_lower(img3, 32)
b4 = bright_lower(img3, 16)
b3 = bright_lower(img3, 8)
```

Рисунок 2. Результат работы функции прореживания квантованием



Задача 2:

В технологии JPEG применяется формат YUV для промежуточного представления изображения. В этом формате два канала (2 и 3) хранят информацию о цвете, а один канал – о яркости. Требуется написать алгоритм прореживания (дэцимации) цветоразностных каналов. Необходимо реализовать отдельно кодирование и декодирование.

Для перевода из RGB в YUV используются формулы:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

$$Cb = -0.1687 * R - 0.3313 * G + 0.5 * B + 128$$

$$Cr = 0.5 * R - 0.4187 * G - 0.0813 * B + 128$$

Для обратного перехода из YUV в RGB:

$$R = Y + 1.402 * (Cr - 128)$$

$$G = Y - 0.34414 * (Cb - 128) - 0.71414 * (Cr - 128)$$

$$B = Y + 1.772 * (Cb - 128)$$

Функции перевода между RGB и YUV

```
def make_u(rgb):
    return (-0.1687*rgb[:, :, 0] - 0.3313*rgb[:, :, 1] + 0.5*rgb[:, :, 2] + 128)
def make_v(rgb):
    return (0.5*rgb[:, :, 0] - 0.4187*rgb[:, :, 1] - 0.0813*rgb[:, :, 2] + 128)
def make_y(rgb):
    return (0.299*rgb[:, :, 0] + 0.587*rgb[:, :, 1] + 0.114*rgb[:, :, 2])
```

В следующих функциях также используется, не выделенный в функцию перевод в `rgb`, так как при разном пересчете выходили разные изображения (см. П1.1)

```
y = make_y(im)
```

```

rd = y[:, :] + 1.402*(avg_v[:, :] - 128)
gr = y[:, :] - 0.34414*(avg_u[:, :] - 128) - 0.71414*(avg_v[:, :] - 128)
bl = y[:, :] + 1.772*(avg_u[:, :] - 128)

```

Функция прореживания цветоразностного канала (Промежуточная)

Рисунок 3. Скриншот алгоритма функции compressF

```

def compressF(m):
    #sizeX = len(m)/2 if len(m)%2 == 0 else (len(m)/2)+1
    #sizeY = len(m[0])/2 if len(m)%2 == 0 else (len(m[0])/2)+1
    #res = np.zeros((sizeX, sizeY))
    res = np.zeros((round(m.shape[0]/2), round(m.shape[1]/2)))
    for i in range (int(round(m.shape[0]/2))):
        for j in range (int(round(m.shape[1]/2))):
            four = []
            four.append(m[i*2, j*2])
            if ((j*2)+1 < m.shape[1]):
                four.append(m[i*2, j*2+1])
            if ((i*2)+1 < m.shape[0]):
                four.append(m[i*2+1, j*2])
            if ((j*2)+1 < m.shape[1] and (i*2)+1 < m.shape[0]):
                four.append(m[i*2+1, j*2+1])
            res[i, j] = sum(four) / len(four)
    return res

```

Проходя по двум циклам с количеством шагов в два раза меньше ширины и длины подаваемого на входе изображения, функция высчитывает среднее значение между значениями в массиве, куда помещаются значения 4x пикселей – по два в строке по два в столбце – подаваемого канала, не выходя за границы массива изображения. На выходе сжатая в 4 раза пурпурная матрица.

Функция развертывания цветоразностного канала (декодирование):

Рисунок 3. Скриншот алгоритма функции uncompressF

```

def uncompressF(m, sizeX, sizeY):
    #array = np.zeros((m.shape[0]*2, m.shape[1]*2))
    array = np.zeros((sizeX*2, sizeY*2))
    for i in range (m.shape[0]):
        for j in range (m.shape[1]):
            a = m[i, j]
            array[i*2+1-1, j*2+1-1] = a
            array[i*2, j*2] = m[i, j]
            #if ((j*2)+1 < sizeX):
            if ((i*2)+1 < sizeX):
                array[i*2+1, j*2] = m[i, j]
            #if ((j*2)+1 < sizeY):
            if ((j*2)+1 < sizeY):
                array[i*2, j*2+1] = m[i, j]
            #if ((i*2)+1 < sizeX & (j*2)+1 < sizeY):
            if ((j*2)+1 < sizeY and (i*2)+1 < sizeX):
                array[i*2+1, j*2+1] = m[i, j]
    return array[0:sizeX, 0:sizeY]

```

Проходя по двойному циклу с количеством шагов в два раза меньше ширины и высоты исходного изображения, происходит развертывание канала – распространение – на 4 пикселя (по два в строке, по два в столбце). На выходе пурпурный массив изображения с размером исходного, чтобы были возможны дальнейшие преобразования попискельно соотносимо с размером массива Y, отвечающего за яркость, который не подвергается сжатию.

Функция децимации цветоразностных каналов изображения:

Рисунок 3. Скриншот алгоритма функции decimationF

```
def decimationF(im):
    half_avg_u = compressF(make_u(im))
    #print(half_avg_u[0:10, 0:10])
    avg_u = uncompressF(half_avg_u, im.shape[0], im.shape[1])
    #print(avg_u)
    half_avg_v = compressF(make_v(im))
    avg_v = uncompressF(half_avg_v, im.shape[0], im.shape[1])
    y = make_y(im)
    rd = y[:, :] + 1.402*(avg_v[:, :] - 128)
    gr = y[:, :] - 0.34414*(avg_u[:, :] - 128) - 0.71414*(avg_v[:, :] - 128)
    bl = y[:, :] + 1.772*(avg_u[:, :] - 128)
    return dstack((rd, gr, bl)).astype('uint8')
```

Последовательно подаются императивы на поканальное сжатие (кодирование) и развертывание (декодирование) полученных при помощи функций make_u и make_v U и V каналов соответственно. Затем происходит выделение без сжатия канала яркости Y и обратный перевод в цветовую модель RGB. На выход подается сформированное из округленных r,gb каналов изображение.

В результате из исходного изображения:

Рисунок 4. Исходное изображение



Выходит:

Рисунок 5. Выходное децимированное изображение



Задача 3:

Для оценки проведенных процедур выполняется

Произвести оценку работы алгоритмов в первых двух заданиях, для этого вычислить энтропию и среднеквадратичную ошибку по формулам:

$$\text{Формула энтропии: } H = -p_i \log p_i \quad (2)$$

$$\text{Формула среднеквадратичной ошибки: } MSE = \frac{1}{n} \sum_{i=1}^n (P_i - Q_i)^2 \quad (3)$$

Любая информация носит емкостный характер. Мера неопределенности информации – энтропия – позволяет количество содержащихся данных в изображении и соответственно сравнивать неопределенность в состоянии представляющей изображение палитры.

В случае с изображением, энтропию можно описать как количественную меру свето-контрастности изображения представленного в градациях серого в первом варианте. Т.е. энтропия изображения как величина, говорит нам насколько много разных интенсивностей может содержать это изображение. Для цветного RGB-изображения поканальная энтропия покажет насколько много разных интенсивностей будет содержать каждый канал изображения, т.е. насколько вариативно будет представление данных в разных его каналах, а среднее значение поканальных энтропий можно интерпретировать как меру информационной наполненности цветного изображения.

Представление энтропии каналов цветного изображения не говорит нам ничего о количественной мере информации как набора тональных оттенков. Для оценки цветовых тонов

можно пересчитать RGB-пространство в удобное для снятия распределения тонов и насыщенностей. Вследствии того, что из цветоразностных сигналов частично исключена избыточная информация о яркости, так как такой сигнал представляет из себя матрицу отделенную от значения яркости, то такой промежуточный канал наиболее всего подходит при измерении энтропии. Учитывая, что выделяются 2 канала и они покажут примерно одинаковое значение разброса, подсчету энтропии изображения будет подлежать только U канал.

Для чистоты измерения каналы будут повторно выделяться из выходных изображений работы функций.

Рисунок 6. Программный код функций подсчета энтропии и среднеквадратичной ошибки

```

def entropy2(one, two):
    sum_one = 0
    sum_two = 0
    size = len(one)*len(one[0])
    for i in range(len(one)):
        for j in range(len(one[0])):
            #pi = math.fabs(one[i,j]-two[i,j])/size
            p1 = (one[i,j]/size)
            p2 = (two[i,j]/size)
            #res += pi* math.log2(pi)
            sum_one += p1* math.log2(p1)
            sum_two += p2* math.log2(p2)
    print(sum_one, sum_two)
    return math.fabs(sum_one - sum_two)

def msi(one, two):
    e = 0
    for i in range(len(one)):
        for j in range(len(one[0])):
            e += round(one[i][j] - two[i][j])**2
    e = (e / (one.shape[0]*one.shape[1] - 1))**(1/2)
    return e

def entropy4(one,two):
    res1 = 0
    res2 = 0
    size = one.shape[0]*one.shape[1]
    values1, bin_edges1, patches1 = hist(one.ravel(), bins=range(257))
    values2, bin_edges2, patches2 = hist(two.ravel(), bins=range(257))
    for i in range(len(values1)):
        p1 = math.fabs(values1[i]/size)
        p2 = math.fabs(values2[i]/size)
        #print(p1, p2)
        if (p1>0):
            res1 += p1* math.log2(p1)
        if (p2>0):
            res2 += p2* math.log2(p2)
    print(res1, res2)
    return math.fabs(res1-res2)

```

3.1. Результаты и пояснение к оценки алгоритма квантования

Рисунок 7. Результаты работы функций для исходной и выходной изображений квантования

```
In [8]: print('\n',msi(img3, b3), '\n',
msi(img3, b4), '\n',
msi(img3, b5), '\n',
msi(img3, b6), '\n',
msi(img3, b7))

/home/nikon-cook/anaconda3/lib/python3.5/s
__py:26: RuntimeWarning: overflow encountered in exp

      132.648509973
      215.489364277
      105.020907975
      110.186426513
      76.864549645

In [30]: entropy4(img3, b3)
-5.2884009483286105 -2.976428755079898

Out[30]: 2.3119721932487125

In [31]: entropy4(img3, b4)
-5.2884009483286105 -2.3543896157492425

Out[31]: 2.934011332579368

In [32]: entropy4(img3, b5)
-5.2884009483286105 -1.8228455423060297

Out[32]: 3.465555406022581

In [33]: entropy4(img3, b6)
-5.2884009483286105 -1.3521708105638073

Out[33]: 3.9362301377648032

In [34]: entropy4(img3, b7)
-5.2884009483286105 -0.6735617455200618

Out[34]: 4.6148392028085485
```

Среднеквадратичная ошибка для изображений достаточно большая для представленных уровней квантования, что не удивительно, так как изображения подвергаются существенному изменению стремя с биноризации изображения. Естественно после некоторого шага изображение будет представлять собой черный квадрат. В данном случае это происходит на шаге 8, при передачи значения квантования $2^8 = 256$.

Энтропию же не получается вычислить по алгоритму entropy2, потому что вероятность появления одного пикселя на изображении получается настолько маленькой, что python не может вычислить log по ней.

Потому написан другой алгоритм entropy4, который вычисляет значение вероятности по гистограммам изображение. Такой способ даже упрощает двойной цикл до одинарного.

Чем выше шаг квантования, тем больше разнятся изображения (как уже сказано ранее), следовательно и энтропия как разница энтропий двух изображений становится больше, что и видно на изображениях.

3.2. Результаты и пояснение к оценки алгоритма децимации

Среднеквадратичная ошибка изображения составила 2.6192239904153247, что понятно при полноразмерном просмотре изображения, подмечаются мелкие детали различий близкие к местам с гранями отличных цветов. Например, как переход между основным фоном и объектом на изображении – монотонные однотипные пиксели резко обращаются в цвет объекта. Но при уменьшении масштаба изображения разница не заметна.



Разница энтропий исходного изображения и выходного (сумма энтропии символов) составляет 2.487972374505034, соотносимо с разницей в сотых долях. Но в данном случае выходит 2 тысячи в сравнении с 4x значными числами. Энтропия исходного изображения: -1509.4025739. Энтропия выходного изображения: -1511.89054627. В таком соотношении разница в однозначное число это как раз немного.

В случае подсчета энтропии по гистограмме тк она фактически отражает частоту появления одного пикселя, функцию можно упростить. Тогда для входного и выходного изображения разница энтропий составит 0.23083322537877926. Соотносимо с входным и выходным значениями энтропий: -4.129718051627284 -4.3605512770060635 – число в десятки раз меньше.

Выводы:

Изображения отличаются друг от друга содержательно, могут иметь определенное количество деталей, и в зависимости от наполнения могут нести разный уровень информативности. При необходимости уменьшить емкость, как объем хранения, возникает вопрос сохранения информативности наряду с сохранением места на физическом носителе. При этом необходимо ввести количественные характеристики изображения, позволяющие оценивать предельные свойства алгоритмов кодирования, исправления и анализа изображений. В данном случае которыми выступают среднеквадратичная ошибка и энтропия. По полученным значениям можно сказать, что алгоритмы, приведенные в данной работе работают достаточно неплохо.

Приложение 1

1.1

Цветоразностный сигнал - цветовой видеосигнал, созданный путем вычитания яркостной и/или цветовой составляющей из одного из первичных цветовых сигналов (красного, зеленого, синего — RGB). В цветоразностном формате Betacam, например, яркость (Y) и цветоразностные компоненты (R-Y и B-Y) соотносятся ранее приведенными формулами.

Цветоразностный сигнал G-V не создается, так как он может быть реконструирован из остальных трех сигналов. Есть и другие соглашения о вычитании цветов: SMPTE, EBU-N1 0 и МП. Цветоразностные сигналы не следует называть компонентами видеосигнала. Этот термин зарезервирован для компонентов RGB. Нестрого термин «компонент видеосигнала» часто используется именно в смысле цветоразностных сигналов.

Формулы определяются с учетом матрицирования (- это операция передачи двух сигналов изображения с восстановлением третьего в приемнике путем вычитания переданных первичных сигналов из сигнала яркости. Операция возможна поскольку вычитание равнозначно сложению инвертированных, т. е. взятых в противоположной полярности сигналов. Во всех системах цветного телевидения принято передавать красный и синий) и в зависимости от спектральной чувствительности зрения настраиваются коэффициенты сигналов.

<http://www.tvgarant.ru/glossary/cvetoraznostnyy-signal>

<http://principact.ru/content/view/92/29/>

1.2

$$r = (0.587 * \text{im}[:, :, 1] + 0.114 * \text{im}[:, :, 2] + 1.402 * (\text{avg_v} - 128)) / (1 - 0.299)$$

$$g = (0.299 * \text{im}[:, :, 0] + 0.114 * \text{im}[:, :, 2] - 0.34414 * (\text{avg_u} - 128) - 0.71414 * (\text{avg_v} - 128)) / (1 - 0.587)$$

$$b = (0.299 * \text{im}[:, :, 0] + 0.587 * \text{im}[:, :, 1] + 1.772 * (\text{avg_u} - 128)) / (1 - 0.114)$$

Дополнительно

Принцип построения телевизионного сигнала

<http://principact.ru/content/view/87/108/>