Video Compression Assignment 3

In this assignment, we will use the block-based encoding approach, where the size of a block is 8x8. Only the Luma component is considered for the following questions.

Prerequisites

- Programming langauge: Python 3.10+ (IPython)
- Framework: Jupyter

Install dependencies from PyPI:

In [1]:
```
%pip install \
    --disable-pip-version-check \
    --quiet \
    numpy \
    Pillow \
    scikit-image
```

```
Note: you may need to restart the kernel to use updated packages.
```

Task 1

> Fourier Transform

Please apply the Fourier Transform to the luma component of `foreman_qcif_0_rgb.bmp` and demonstrate its magnitudes in a 2-D image, as shown in the example below. Note that you need to shift the origin to the center of the image for the magnitude plot.

Solution of Task 1

In [ ]:

```

```

Task 2

> DCT

Please apply DCT to all the 8x8 luma blocks of `foreman_qcif_0_rgb.bmp` and use the quantization matrix below for quantization. After DCT and quantization, please apply inverse quantization and IDCT to decode all the blocks and show the decoded frame.

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Solution of Task 2

2-D Discrete Cosine Transform

Forward Transform (for `NxN` block)

The formula adapted from the course slides:

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\left(\frac{(2x+1)u\pi}{2N}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right)$$

$$\text{where } C(t) = \begin{cases} \frac{2}{\sqrt{N}} & \text{if } t = 0 \\ 2\sqrt{\frac{2}{N}} & \text{otherwise} \end{cases}$$

can be simplified as:

$$F(u,v) = \frac{8}{N^2} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\left(\frac{(2x+1)u\pi}{2N}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right)$$

$$\text{where } C(t) = \begin{cases} 1 & \text{if } t = 0 \\ \sqrt{2} & \text{otherwise} \end{cases}$$

Inverse Transform (for `NxN` block)

The formula adapted from the course slides:

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\left(\frac{(2x+1)u\pi}{2N}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right)$$

$$\text{where } C(t) = \begin{cases} \frac{2}{\sqrt{N}} & \text{if } t = 0 \\ 2\sqrt{\frac{2}{N}} & \text{otherwise} \end{cases}$$

can be simplified as:

$$f(x,y) = \frac{8}{N^2} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\left(\frac{(2x+1)u\pi}{2N}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right)$$

$$\text{where } C(t) = \begin{cases} 1 & \text{if } t = 0 \\ \sqrt{2} & \text{otherwise} \end{cases}$$

Implement the DCT and IDCT functions:

In [2]:

```python
from numpy.typing import NDArray
from numpy import float64, uint8


def dct_forward(source: NDArray[uint8], block_size: int) -> NDArray[float64]:
    from numpy import atleast_2d, float64
    from math import sqrt, cos, pi

    source = atleast_2d(source).astype(float64)
    block_size = int(block_size)

    N = block_size
    C = 8 / N / N
```

```python
    PI_2N = pi / 2 / N
    SR_2 = sqrt(2)

    target = source.copy()
    for i in range(0, target.shape[0] - N + 1, N):
        for j in range(0, target.shape[1] - N + 1, N):
            for u in range(N):
                for v in range(N):
                    a = 0.0
                    for x in range(N):
                        for y in range(N):
                            a += (
                                source[i + x, j + y]
                                * cos((2 * x + 1) * u * PI_2N)
                                * cos((2 * y + 1) * v * PI_2N)
                            )
                    target[i + u, j + v] = (
                        C
                        * (1 if u == 0 else SR_2)
                        * (1 if v == 0 else SR_2)
                        * a
                    )
    return target


def dct_inverse(source: NDArray[float64], block_size: int) -> NDArray[uint8]:
    from numpy import atleast_2d, float64, uint8
    from math import cos, pi, sqrt

    source = atleast_2d(source).astype(float64)
    block_size = int(block_size)

    N = block_size
    C = 8 / N / N
    PI_2N = pi / 2 / N
    SR_2 = sqrt(2)

    target = source.copy()
    for i in range(0, target.shape[0] - N + 1, N):
        for j in range(0, target.shape[1] - N + 1, N):
            for x in range(N):
                for y in range(N):
                    a = 0.0
                    for u in range(N):
                        for v in range(N):
                            a += (
                                (1 if u == 0 else SR_2)
                                * (1 if v == 0 else SR_2)
                                * source[i + u, j + v]
                                * cos((2 * x + 1) * u * PI_2N)
                                * cos((2 * y + 1) * v * PI_2N)
                            )
                    target[i + x, j + y] = C * a
    target = target.round().clip(max=255).astype(uint8)
    return target
```

In [3]:

```python
from numpy import set_printoptions

set_printoptions(suppress=True, precision=2)
```

Test the DCT and IDCT functions:

In [4]:
```python
from numpy.random import randint

mat = randint(low=0, high=256, size=(17, 8), dtype=uint8)
mat_dct = dct_forward(mat, 8)
mat_idct = dct_inverse(mat_dct, 8)

print(mat_dct)

assert (mat == mat_idct).all()
```

```
[[ 859.88  -27.18  187.6    55.58  -79.37  -18.47   58.    -18.67]
 [ -75.65  -95.07 -100.51  103.21   11.99  -26.68  101.69  -73.74]
 [-128.37  -38.05   22.04  -29.45   41.5    32.43   35.33  -29.86]
 [ -42.35  117.38  -75.88 -109.37  -89.67   34.17   19.91  -33.38]
 [  -3.12  -82.58    6.69   82.54   12.62  -40.79  -96.53  -27.11]
 [ -47.11  106.36  -54.53   81.44 -113.15  -27.79   -3.86   50.55]
 [-158.03  -66.13   62.08 -119.2    77.65   46.62   32.46   55.87]
 [ -31.72  156.53  121.03   64.5    91.36   10.73   45.02  -94.77]
 [1013.38    9.13 -196.31    0.1   -29.12  -78.27   99.31   44.27]
 [ -45.71  -55.28  -10.49   41.4  -121.37   -5.73   -3.31  -82.17]
 [ -16.03    0.13  -72.63  -42.23   42.85   44.51  -17.85 -163.05]
 [  49.27 -144.38    5.44   98.52   76.11   10.25  -80.17  106.49]
 [ -71.13  -16.17 -151.68   79.44   29.88  -32.2    -1.98   28.3 ]
 [  -8.76   20.49   -9.16   90.89  190.76   -2.1    67.34    0.14]
 [  -7.02  -35.14  -14.85  -21.35  -59.55   27.33  -98.62   33.96]
 [  17.58  -15.02   12.23 -127.29   88.7    78.92   57.08    8.86]
 [  99.    205.    117.    226.    108.    204.    184.     45.  ]]
```

Quantization

Forward Transform

$$F'(u,v) = \text{round}\left(\frac{F(u,v)}{Q(u,v)}\right) \text{ where } Q \text{ is the quantization matrix}$$

Inverse Transform

$$F(u,v) = F'(u,v) \cdot Q(u,v) \text{ where } Q \text{ is the quantization matrix}$$

In [5]:
```python
from numpy.typing import NDArray
from numpy import array, float64, int64, uint8
from typing import Optional

COMMON_QUANTIZATION_MATRIX = array(
    [
        [16, 11, 10, 16, 24, 40, 51, 61],
        [12, 12, 14, 19, 26, 58, 60, 55],
        [14, 13, 16, 24, 40, 57, 69, 56],
        [14, 17, 22, 29, 51, 87, 80, 62],
        [18, 22, 37, 56, 68, 109, 103, 77],
        [24, 35, 55, 64, 81, 104, 113, 92],
        [49, 64, 78, 87, 103, 121, 120, 101],
        [72, 92, 95, 98, 112, 100, 103, 99],
    ],
    dtype=uint8,
)


def quantize_forward(
    source: NDArray[float64],
```

```python
    matrix: Optional[NDArray[uint8]] = None,
) -> NDArray[int64]:
    from numpy import atleast_2d, float64, uint8, int64

    source = atleast_2d(source).astype(float64)
    matrix = (
        atleast_2d(matrix).astype(uint8)
        if matrix
        else COMMON_QUANTIZATION_MATRIX
    )

    target = source.copy()
    for i in range(0, target.shape[0] - matrix.shape[0] + 1, matrix.shape[0]):
        for j in range(
            0, target.shape[1] - matrix.shape[1] + 1, matrix.shape[1]
        ):
            target[i : i + matrix.shape[0], j : j + matrix.shape[1]] = (
                source[i : i + matrix.shape[0], j : j + matrix.shape[1]]
                / matrix
            )
    target = target.round().astype(int64)
    return target


def quantize_inverse(
    source: NDArray[int64],
    matrix: Optional[NDArray[uint8]] = None,
) -> NDArray[float64]:
    from numpy import atleast_2d, float64, uint8

    source = atleast_2d(source).astype(float64)
    matrix = (
        atleast_2d(matrix).astype(uint8)
        if matrix
        else COMMON_QUANTIZATION_MATRIX
    )

    target = source.copy()
    for i in range(0, target.shape[0] - matrix.shape[0] + 1, matrix.shape[0]):
        for j in range(
            0, target.shape[1] - matrix.shape[1] + 1, matrix.shape[1]
        ):
            target[i : i + matrix.shape[0], j : j + matrix.shape[1]] = (
                source[i : i + matrix.shape[0], j : j + matrix.shape[1]]
                * matrix
            )
    return target
```

Test the quantization and inverse quantization functions:

In [6]:

```python
from numpy.random import randn
from numpy import set_printoptions

mat = randn(17, 8) * 100
mat_q = quantize_forward(mat)
mat_iq = quantize_inverse(mat_q)

set_printoptions(suppress=True)

print(abs((mat - mat_iq)).mean())
print(mat_q)
```

```
13.294140104672888
[[   5  -11   16   -8    6    0   -2    1]
 [   0    4    3    4    5    0   -2    0]
 [   8    0   -5    0   -1    1    2   -2]
 [  -1   -1   -1   -4   -2   -1    0    0]
 [   3    4   -1    0    0   -1    0   -2]
 [   6    4   -1   -2    2    0    0    0]
 [   0    0    1    0    0    0    1   -2]
 [  -2    0    0    4    0    1    2    1]
 [   5   10    7   12    0    0   -1   -1]
 [ -10    1    2   -5   -1   -1    1    1]
 [   5    1   -6    0    0    2    0    3]
 [   4   -4    0    2    0   -2   -1    0]
 [   5   -4   -1    0   -2    1    0   -1]
 [  -8    1    1    2    0    0    0   -1]
 [  -1    3   -1    2    0    0    2   -1]
 [  -1    0    0    0    0    2   -1   -1]
 [   9  -52   33   26  -20 -188 -118 -125]]
```

Load the image and apply the DCT and quantization:

In [7]:
```
from IPython.display import display, Markdown, Image as I
from pathlib import Path
from PIL import Image
from numpy import asarray
from skimage.metrics import structural_similarity as ssim

source_path = Path("../resources/foreman_qcif_0_rgb.bmp").resolve()
source_image = Image.open(source_path).convert(mode="L")
source_path = Path("images/foreman_qcif_0_rgb.luma.source.bmp").resolve()
source_data = asarray(source_image)
transformed_data = dct_forward(source_data, 8)
quantized_data = quantize_forward(transformed_data)
dequantized_data = quantize_inverse(quantized_data)
target_data = dct_inverse(dequantized_data, 8)
target_image = Image.fromarray(target_data, mode="L")
target_path = Path("images/foreman_qcif_0_rgb.luma.target.bmp").resolve()
fidelity_ssim = ssim(source_data, target_data)

table_view = f"""\
| Variable Name | Value |
|---------------|-------|
| source_image  | ![source_image]({source_path}) |
| target_image  | ![target_image]({target_path}) |
| fidelity_ssim | `{fidelity_ssim:.4f}` |
"""

for var_name in [
    "source_data",
    "transformed_data",
    "quantized_data",
    "dequantized_data",
    "target_data",
]:
    var_value = locals()[var_name]
    table_view += (
        f"| {var_name} | `{str(var_value).replace("\n", "`<br>`")}` |\n"
    )

source_image.save(source_path)
target_image.save(target_path)
display(Markdown(table_view))
```

Variable Name	Value
source_image	
target_image	
fidelity_ssim	0.9418
source_data	[[32 233 251 ... 212 230 203] [39 212 206 ... 220 226 203] [37 207 187 ... 191 228 200] ... [14 132 215 ... 176 174 154] [15 132 215 ... 191 193 178] [14 131 212 ... 128 127 118]]
transformed_data	[[1531.5 -326.07 -276.39 ... 19.8 -16.01 1.28] [24.96 96.18 11.23 ... 10.26 -6.66 7.32] [74.87 -58.06 -38.65 ... -9.62 -2.91 -2.16] ... [-1.49 1.55 1.83 ... 5.95 8.25 5.29] [-1.68 1.61 -2.55 ... 11.64 0.76 0.98] [2.79 -1.92 -0.82 ... -0.21 -5.4 -1.23]]
quantized_data	[[96 -30 -28 ... 0 0 0] [2 8 1 ... 0 0 0] [5 -4 -2 ... 0 0 0] ... [0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]]
dequantized_data	[[1536. -330. -280. ... 0. 0. 0.] [24. 96. 14. ... 0. 0. 0.] [70. -52. -32. ... 0. 0. 0.] ... [0. 0. 0. ... 0. 0. 0.] [0. 0. 0. ... 0. 0. 0.] [0. 0. 0. ... 0. 0. 0.]]
target_data	[[22 242 255 ... 222 215 207] [43 203 202 ... 220 224 204] [55 200 196 ... 189 219 215] ... [14 133 213 ... 179 169 141] [13 132 211 ... 188 194 194] [13 132 211 ... 132 121 114]]