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

2D Discrete Fourier Transform

Forward Transform

The formula adapted from the course slides is as follows:

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-i2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$
$$\text{where } u = 0,1,\ldots,M-1 \text{ and } v = 0,1,\ldots,N-1$$

Origin Shift

Move all the pixels horizontally and vertically by half of the image width and height, respectively:

$$F'(u,v) = F((u+\lfloor M/2\rfloor) \bmod M, (v+\lfloor N/2\rfloor) \bmod N)$$
$$\text{where } u = 0,1,\ldots,M-1 \text{ and } v = 0,1,\ldots,N-1$$

Reduced Magnitude

The reduced magnitude of the Fourier Transform is calculated as follows:

$$G(u,v) = \log(1+|F'(u,v)|)$$
$$\text{where } u = 0,1,\ldots,M-1 \text{ and } v = 0,1,\ldots,N-1$$

Normalization

The magnitude values are normalized to the range `[0, 255]` using the following formula:

$$G'(u,v) = 0 + 255 \times \frac{G(u,v)-\min(G)}{\max(G)-\min(G)}$$
$$\text{where } u = 0,1,\ldots,M-1 \text{ and } v = 0,1,\ldots,N-1$$

Implement the above steps in the following code:

In [2]:
```
from numpy.typing import NDArray
from numpy import complex128, float64, uint8


def dft_forward(source: NDArray[uint8]) -> NDArray[complex128]:
    from numpy import arange, atleast_2d, exp, meshgrid, pi, uint8

    source = atleast_2d(source).astype(uint8)

    M = source.shape[0]
    N = source.shape[1]
    C = 1 / M / N
    NEG_I2PI = -2j * pi
    X, Y = meshgrid(arange(M), arange(N), indexing="ij")
    X_M = X / M
    Y_N = Y / N

    target = source.copy().astype(complex128)
    for u in range(M):
        for v in range(N):
            target[u, v] = (
                C * (source * exp(NEG_I2PI * u * X_M + NEG_I2PI * v *
```

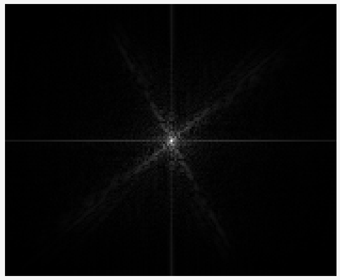
```
Y_N)).sum()
            )
    return target


def planar_shift(source: NDArray) -> NDArray:
    from numpy import atleast_2d, roll

    source = atleast_2d(source)

    M = source.shape[0]
    N = source.shape[1]
    M_2 = M // 2
    N_2 = N // 2

    target = roll(source, shift=(M_2, N_2), axis=(0, 1))
    return target


def log_scale(source: NDArray[complex128]) -> NDArray[float64]:
    from numpy import asarray, complex128, log1p

    source = asarray(source).astype(complex128)

    target = log1p(abs(source))
    return target


def norm_scale(source: NDArray[float64]) -> NDArray[uint8]:
    from numpy import asarray, float64

    source = asarray(source).astype(float64)

    scaling = 255 / (source.max() - source.min())
    target = (scaling * (source - source.min())).round().astype(uint8)
    return target
```

Load the image, apply the Fourier Transform to the luma component, and demonstrate its magnitudes in a 2-D image:

In [3]:

```
from IPython.display import display, Markdown
from pathlib import Path
from PIL import Image
from numpy import asarray, set_printoptions

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 = dft_forward(source_data)
shifted_data = planar_shift(transformed_data)
target_data = norm_scale(log_scale(shifted_data))
target_image = Image.fromarray(target_data, mode="L")
target_path = Path("images/foreman_qcif_0_rgb.luma.target.1.bmp").resolve()

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

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

for var_name in [
    "source_data",
    "transformed_data",
    "shifted_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	

source_data	`[[ 32 233 ... 230 203]`
	`[ 39 212 ... 226 203]`
	`...`
	`[ 15 132 ... 193 178]`
	`[ 14 131 ... 127 118]]`

transformed_data	`[[169.22 +0.j    9.8  -7.31j ...   1.95 +4.58j   9.8  +7.31j]`
	`[  5.46-19.69j  -3.25 -0.39j ...  -3.68 +6.36j  -9.16+11.63j]`
	`...`
	`[ -4.03 +9.06j  -1.66 -1.36j ...   0.62 +3.52j   9.53 -6.11j]`
	`[  5.46+19.69j  -9.16-11.63j ...   4.06 +0.47j  -3.25 +0.39j]]`

shifted_data	`[[-0.  -0.j -0.  -0.j ... -0.  -0.j -0.  +0.j]`
	`[-0.  -0.j -0.  +0.j ... -0.  -0.j -0.  -0.j]`
	`...`
	`[-0.  +0.j -0.  +0.j ... -0.  -0.j -0.01-0.j]`
	`[-0.  +0.j -0.  +0.j ... -0.  -0.j -0.  -0.j]]`

target_data	`[[0 0 ... 0 0]`
	`[0 0 ... 0 0]`
	`...`
	`[0 0 ... 0 0]`
	`[0 0 ... 0 0]]`

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:

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

$$\text{where } C(t) = \begin{cases} \frac{2}{\sqrt{N}} & \text{if } t = 0 \\ 2\sqrt{\frac{2}{N}} & \text{otherwise} \end{cases} \quad \text{and } u, v = 0, 1, \ldots, N-1$$

can be simplified as:

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

$$\text{where } C(t) = \begin{cases} 1 & \text{if } t = 0 \\ \sqrt{2} & \text{otherwise} \end{cases} \quad \text{and } u, v = 0, 1, \ldots, N-1$$

Inverse Transform (for `NxN` block)

The formula adapted from the course slides:

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

$$\text{where } C(t) = \begin{cases} \frac{2}{\sqrt{N}} & \text{if } t = 0 \\ 2\sqrt{\frac{2}{N}} & \text{otherwise} \end{cases} \quad \text{and } u, v = 0, 1, \ldots, N-1$$

can be simplified as:

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

$$\text{where } C(t) = \begin{cases} 1 & \text{if } t = 0 \\ \sqrt{2} & \text{otherwise} \end{cases} \quad \text{and } u, v = 0, 1, \ldots, N-1$$

Implement the DCT and IDCT functions:

In [4]:

```
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, uint8
    from math import cos, pi

    source = atleast_2d(source).astype(uint8)
    block_size = int(block_size)

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

```python
    PI_2N = pi / 2 / N
    SR_2 = 2 ** 0.5

    target = source.copy().astype(float64)
    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

    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 = 2 ** 0.5

    target = source.copy().astype(uint8)
    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] = round(min(C * a, 255))
    return target
```

Test the DCT and IDCT functions:

In [5]:
```python
from numpy.random import randint

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

```
print(mat_dct)

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

```
[[ 998.62 -101.47  -73.      9.92  -45.37  103.65   61.22   53.05]
 [-123.6    32.45   49.03  -27.76    4.69 -110.83   36.63  -45.25]
 [-111.9    11.71   94.41   40.94  -76.9     3.15  -82.69   58.89]
 [-116.17 -160.62   74.87  -89.07  -58.19   35.51  -15.16  -74.8 ]
 [ 118.62  -84.47   24.69  -41.98   -5.38  -10.03  115.47   37.76]
 [   2.6   -51.38   13.38   -8.19  -12.99  -90.97   35.45 -108.76]
 [  69.22   90.81  -23.19   -8.55   84.48   89.35   14.34  -42.14]
 [ -38.05  -81.45  -38.25   36.47   78.4   -92.07  -25.09  -24.4 ]]
```

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 [6]:

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

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

    target = source.copy().astype(int64)
    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
            ).round()
    return target


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

    source = atleast_2d(source).astype(int64)
    matrix = (
        atleast_2d(matrix).astype(uint8)
        if matrix
        else COMMON_QUANTIZATION_MATRIX
    )

    target = source.copy().astype(float64)
    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 [7]:
```
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)
```

```
12.949407959340158
[[ 12   4 -17  -2   4  -1   3   1]
 [ -5  -2  -3  -1  -4  -2   0  -1]
 [ -3  -4   0   7   0   0   1   2]
 [  1  -4   2   5  -4   0   1   1]
 [  5   6   4   0  -2   1  -1   1]
 [ -1  -4   1   1   0   1   0   1]
 [  3   1  -2   0   1   1   1   1]
 [  1   0   0   0   1   0  -1   0]
 [  0   8  -9  13   0   1  -1  -1]
 [ -5 -11  10  -3   1  -2  -1   3]
 [  2   1  14  -2  -2  -2   1   2]
 [ 10   6  -1   0   1   0   0  -6]
 [  2   1   3   2   2   0   0   1]
 [ -1   0   1   3  -2   1   0   0]
 [  2   0  -1  -1   0   0   2   0]
 [  0  -1   2  -2   0   1  -1   1]
 [ 20  20 -24 183  67 193 -60 -19]]
```

Load the image and apply the DCT, quantization, inverse quantization, and IDCT functions to all the 8x8 blocks:

In [8]:

```python
from IPython.display import display, Markdown
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.2.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 ... 230 203]`
	` [ 39 212 ... 226 203]`
	` ...`
	` [ 15 132 ... 193 178]`
	` [ 14 131 ... 127 118]]`

transformed_data	`[[1531.5  -326.07 ...  -16.01    1.28]`
	` [  24.96   96.18 ...   -6.66    7.32]`
	` ...`
	` [  -1.68    1.61 ...    0.76    0.98]`
	` [   2.79   -1.92 ...   -5.4    -1.23]]`

quantized_data	`[[ 96 -30 ...   0   0]`
	` [  2   8 ...   0   0]`
	` ...`
	` [  0   0 ...   0   0]`
	` [  0   0 ...   0   0]]`

dequantized_data	`[[1536. -330. ...    0.    0.]`
	` [  24.   96. ...    0.    0.]`
	` ...`
	` [   0.    0. ...    0.    0.]`
	` [   0.    0. ...    0.    0.]]`

target_data	`[[ 22 242 ... 215 207]`
	` [ 43 203 ... 224 204]`
	` ...`
	` [ 13 132 ... 194 194]`
	` [ 13 132 ... 121 114]]`