encoder.py

```python
from math import ceil
import cv2
import numpy as np
from PIL import Image
from pathlib import Path

from utils import *
from huffman import *


def padding(im, mh, mw):
    """
    pad use boundary pixels so that its height and width are
    the multiple of the height and width of MCUs, respectively
    """
    h, w, d = im.shape
    if h % mh == 0 and w % mw == 0:
        return im
    hh, ww = ceil(h / mh) * mh, ceil(w / mw) * mw
    im_ex = np.zeros_like(im, shape=(hh, ww, d))
    im_ex[:h, :w] = im
    im_ex[:, w:] = im_ex[:, w - 1 : w]
    im_ex[h:, :] = im_ex[h - 1 : h, :]
    return im_ex


mcu_sizes = {
    "4:2:0": (BH * 2, BW * 2),
    "4:1:1": (BH * 2, BW * 2),
    "4:2:2": (BH, BW * 2),
    "4:4:4": (BH, BW),
}


def scan_blocks(mcu, mh, mw):
    """
```

```
    scan MCU to blocks for DPCM, for 4:2:0, the scan order is as follows:
    --------- | ---------
    | 0 | 1 | | | 4 | 5 |
    --------- | ---------
    | 2 | 3 | | | 6 | 7 |
    --------- | ---------
    """
    blocks = (
        mcu.reshape(-1, mh // BH, BH, mw // BW, BW).swapaxes(2, 3).reshape(-1, BH, BW)
    )
    return blocks


def DCT(blocks):
    dct = np.zeros_like(blocks)
    for i in range(blocks.shape[0]):
        dct[i] = cv2.dct(blocks[i])
    return dct


def zigzag_encode(dct):
    trace = zigzag_points(BH, BW)
    zz = np.zeros_like(dct).reshape(-1, BH * BW)
    for i, p in enumerate(trace):
        zz[:, i] = dct[:, p[0], p[1]]
    return zz


def quantization(dct, table):
    ret = dct / table[None]
    return np.round(ret).astype(np.int32)


def DPCM(dct):
    """
    encode the DC differences
    """
    dc_pred = dct.copy()
    dc_pred[1:, 0] = dct[1:, 0] - dct[:-1, 0]
```

```
    return dc_pred


def run_length_encode(arr):
    # determine where the sequence is ending prematurely
    last_nonzero = -1
    for i, elem in enumerate(arr):
        if elem != 0:
            last_nonzero = i
    rss, values = [], []
    run_length = 0
    for i, elem in enumerate(arr):
        if i > last_nonzero:
            rss.append(0)
            values.append(0)
            break
        elif elem == 0 and run_length < 15:
            run_length += 1
        else:
            size = bits_required(elem)
            rss.append((run_length << 4) + size)
            values.append(elem)
            run_length = 0
    return rss, values


def encode_header(qts, hts, cop_infos, height, width):
    writer = BytesWriter()
    add_bytes = writer.add_bytes
    add_bytes(
        MARKER.SOI,
        MARKER.APP0,
        b"\x00\x10",  # length = 16
        b"JFIF\x00",  # identifier = JFIF0
        b"\x01\x01",  # version
        b"\x00",  # unit
        b"\x00\x01",  # x density
        b"\x00\x01",  # y density
        b"\x00\x00",  # thumbnail data
```

```
    )
    for id_, qt in enumerate(qts):
        add_bytes(
            MARKER.DQT,
            b"\x00C",  # length = 67
            # precision (8 bits), table id, = 0, id_
            int2bytes(id_, 1),
            qt.astype(np.uint8).tobytes(),
        )
    cop_num = len(cop_infos)
    add_bytes(
        MARKER.SOF0,
        int2bytes(8 + 3 * cop_num, 2),  # length
        int2bytes(8, 1),  # 8 bit precision
        int2bytes(height, 2),
        int2bytes(width, 2),
        int2bytes(cop_num, 1),
    )
    add_bytes(*[info.encode_SOF0_info() for info in cop_infos])

    # type << 4 + id, (type 0: DC, 1 : AC)
    type_ids = [b"\x00", b"\x10", b"\x01", b"\x11"]
    for type_id, ht in zip(type_ids, hts):
        count, weigh = convert_huffman_table(ht)
        ht_bytes = count.tobytes() + weigh.tobytes()
        add_bytes(
            MARKER.DHT,
            int2bytes(len(ht_bytes) + 3, 2),  # length
            type_id,
            ht_bytes,
        )

    add_bytes(
        MARKER.SOS,
        int2bytes(6 + cop_num * 2, 2),  # length
        int2bytes(cop_num, 1),
    )
    add_bytes(*[info.encode_SOS_info() for info in cop_infos])
    add_bytes(b"\x00\x3f\x00")
```

```
154     return writer
155
156
157 def encode_mcu(mcu, hts):
158     bit_stream = BitStreamWriter()
159     for cur in mcu:
160         for dct, (dc_ht, ac_ht) in zip(cur, hts):
161             dc_code = encode_2s_complement(dct[0])
162             container = [dc_ht[len(dc_code)],
dc_code]
163             rss, values = run_length_encode(dct[1:])
164             for rs, v in zip(rss, values):
165                 container.append(ac_ht[rs])
166
container.append(encode_2s_complement(v))
167             bitstring = "".join(container)
168             bit_stream.write_bitstring(bitstring)
169     return bit_stream.to_bytes()
170
171
172 def encode_jpeg(im, quality=95, subsample="4:2:0",
use_rm_ht=True):
173     im = np.expand_dims(im, axis=-1) if im.ndim == 2
else im
174     height, width, depth = im.shape
175
176     mh, mw = mcu_sizes[subsample] if depth == 3 else
(BH, BW)
177     im = padding(im, mh, mw)
178     im = RGB2YCbCr(im) if depth == 3 else im
179
180     # DC level shift for luminance,
181     # the shift of chroma was completed by color
conversion
182     Y_im = im[:, :, 0] - 128
183     # divide image into MCUs
184     mcu = divide_blocks(Y_im, mh, mw)
185     # MCU to blocks, for luminance there are more
than one blocks in each MCU
186     Y = scan_blocks(mcu, mh, mw)
187     Y_dct = DCT(Y)
188     # the quantization table was already processed by
zigzag scan,
```

```
189     # so we apply zigzag encoding to DCT block first
190     Y_z = zigzag_encode(Y_dct)
191     qt_y = load_quantization_table(quality, "lum")
192     Y_q = quantization(Y_z, qt_y)
193     Y_p = DPCM(Y_q)
194     # whether to use recommended huffman table
195     if use_rm_ht:
196         Y_dc_ht, Y_ac_ht = reverse(RM_Y_DC),
    reverse(RM_Y_AC)
197     else:
198         Y_dc_ht =
    jpegCreateHuffmanTable(np.vectorize(bits_required)
    (Y_p[:, 0]))
199         Y_ac_ht = jpegCreateHuffmanTable(
200             flatten(run_length_encode(Y_p[i, 1:])[0]
    for i in range(Y_p.shape[0]))
201         )
202     qts, hts = [qt_y], [Y_dc_ht, Y_ac_ht]
203     cop_infos = [ComponentInfo(1, mw // BW, mh // BH,
    0, 0, 0)]
204     # the number of Y DCT blocks in an MCU
205     num = (mw // BW) * (mh // BH)
206     mcu_hts = [(Y_dc_ht, Y_ac_ht) for _ in
    range(num)]
207     # assign DCT blocks to MCUs
208     mcu_ = Y_p.reshape(-1, num, BH * BW)
209
210     if depth == 3:
211         # chroma subsample
212         ch = im[:: mh // BH, :: mw // BW, 1:]
213         Cb = divide_blocks(ch[:, :, 0], BH, BW)
214         Cr = divide_blocks(ch[:, :, 1], BH, BW)
215         Cb_dct, Cr_dct = DCT(Cb), DCT(Cr)
216         Cb_z, Cr_z = zigzag_encode(Cb_dct),
    zigzag_encode(Cr_dct)
217         qt_c = load_quantization_table(quality,
    "chr")
218         Cb_q, Cr_q = quantization(Cb_z, qt_c),
    quantization(Cr_z, qt_c)
219         Cb_p, Cr_p = DPCM(Cb_q), DPCM(Cr_q)
220         if use_rm_ht:
221             C_dc_ht, C_ac_ht = reverse(RM_C_DC),
    reverse(RM_C_AC)
```

```
222         else:
223             ch_ = np.concatenate([Cb_p, Cr_p],
    axis=0)
224             C_dc_ht =
    jpegCreateHuffmanTable(np.vectorize(bits_required)
    (ch_[:, 0]))
225             C_ac_ht = jpegCreateHuffmanTable(
226                 flatten(run_length_encode(ch_[i, 1:])
    [0] for i in range(ch_.shape[0]))
227             )
228         qts.append(qt_c), hts.extend([C_dc_ht,
    C_ac_ht])
229         cop_infos.extend(
230             [ComponentInfo(2, 1, 1, 1, 1, 1),
    ComponentInfo(3, 1, 1, 1, 1, 1)]
231         )
232         mcu_hts.extend((C_dc_ht, C_ac_ht) for _ in
    range(2))
233         mcu_ = np.concatenate([mcu_, Cb_p[:, None],
    Cr_p[:, None]], axis=1)
234
235     writer = encode_header(qts, hts, cop_infos,
    height, width)
236     bytes_ = encode_mcu(mcu_, mcu_hts)
237     writer.write(bytes_.replace(b"\xff",
    b"\xff\x00"))
238     writer.write(MARKER.EOI)
239     return writer.getvalue()
240
241
242 def write_jpeg(filename, im, quality=95,
    subsample="4:2:0", use_rm_ht=True):
243     bytes_ = encode_jpeg(im, quality, subsample,
    use_rm_ht)
244     Path(filename).write_bytes(bytes_)
245
246
247 def main():
248     im = Image.open("./data/villeLyon.jpg")
249     write_jpeg("data/villeLyonLow.jpg", np.array(im),
    5, "4:1:1", False)
250
251
252 if __name__ == "__main__":
```

```
253      main()
```