#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import csv

import copy

import argparse

import itertools

from collections import Counter

from collections import deque

import cv2 as cv

import numpy as np

import mediapipe as mp

from utils import CvFpsCalc

from model import KeyPointClassifier

from model import PointHistoryClassifier

def get\_args():

    parser = argparse.ArgumentParser()

    parser.add\_argument("--device", type=int, default=0)

    parser.add\_argument("--width", help='cap width', type=int, default=960)

    parser.add\_argument("--height", help='cap height', type=int, default=540)

    parser.add\_argument('--use\_static\_image\_mode', action='store\_true')

    parser.add\_argument("--min\_detection\_confidence",

                        help='min\_detection\_confidence',

                        type=float,

                        default=0.7)

    parser.add\_argument("--min\_tracking\_confidence",

                        help='min\_tracking\_confidence',

                        type=int,

                        default=0.5)

    args = parser.parse\_args()

    return args

def main():

    # Argument parsing #################################################################

    args = get\_args()

    cap\_device = args.device

    cap\_width = args.width

    cap\_height = args.height

    use\_static\_image\_mode = args.use\_static\_image\_mode

    min\_detection\_confidence = args.min\_detection\_confidence

    min\_tracking\_confidence = args.min\_tracking\_confidence

    use\_brect = True

    # Camera preparation ###############################################################

    cap = cv.VideoCapture(cap\_device)

    cap.set(cv.CAP\_PROP\_FRAME\_WIDTH, cap\_width)

    cap.set(cv.CAP\_PROP\_FRAME\_HEIGHT, cap\_height)

    # Model load #############################################################

    mp\_hands = mp.solutions.hands

    hands = mp\_hands.Hands(

        static\_image\_mode=use\_static\_image\_mode,

        max\_num\_hands=1,

        min\_detection\_confidence=min\_detection\_confidence,

        min\_tracking\_confidence=min\_tracking\_confidence,

    )

    keypoint\_classifier = KeyPointClassifier()

    point\_history\_classifier = PointHistoryClassifier()

    # Read labels ###########################################################

    with open('model/keypoint\_classifier/keypoint\_classifier\_label.csv',

              encoding='utf-8-sig') as f:

        keypoint\_classifier\_labels = csv.reader(f)

        keypoint\_classifier\_labels = [

            row[0] for row in keypoint\_classifier\_labels

        ]

    with open(

            'model/point\_history\_classifier/point\_history\_classifier\_label.csv',

            encoding='utf-8-sig') as f:

        point\_history\_classifier\_labels = csv.reader(f)

        point\_history\_classifier\_labels = [

            row[0] for row in point\_history\_classifier\_labels

        ]

    # FPS Measurement ########################################################

    cvFpsCalc = CvFpsCalc(buffer\_len=10)

    # Coordinate history #################################################################

    history\_length = 16

    point\_history = deque(maxlen=history\_length)

    # Finger gesture history ################################################

    finger\_gesture\_history = deque(maxlen=history\_length)

    #  ########################################################################

    mode = 0

    while True:

        fps = cvFpsCalc.get()

        # Process Key (ESC: end) #################################################

        key = cv.waitKey(10)

        if key == 27:  # ESC

            break

        number, mode = select\_mode(key, mode)

        # Camera capture #####################################################

        ret, image = cap.read()

        if not ret:

            break

        image = cv.flip(image, 1)  # Mirror display

        debug\_image = copy.deepcopy(image)

        # Detection implementation #############################################################

        image = cv.cvtColor(image, cv.COLOR\_BGR2RGB)

        image.flags.writeable = False

        results = hands.process(image)

        image.flags.writeable = True

        #  ####################################################################

        if results.multi\_hand\_landmarks is not None:

            for hand\_landmarks, handedness in zip(results.multi\_hand\_landmarks,

                                                  results.multi\_handedness):

                # Bounding box calculation

                brect = calc\_bounding\_rect(debug\_image, hand\_landmarks)

                # Landmark calculation

                landmark\_list = calc\_landmark\_list(debug\_image, hand\_landmarks)

                # Conversion to relative coordinates / normalized coordinates

                pre\_processed\_landmark\_list = pre\_process\_landmark(

                    landmark\_list)

                pre\_processed\_point\_history\_list = pre\_process\_point\_history(

                    debug\_image, point\_history)

                # Write to the dataset file

                logging\_csv(number, mode, pre\_processed\_landmark\_list,

                            pre\_processed\_point\_history\_list)

                # Hand sign classification

                hand\_sign\_id = keypoint\_classifier(pre\_processed\_landmark\_list)

                if hand\_sign\_id == 2:  # Point gesture

                    point\_history.append(landmark\_list[8])

                else:

                    point\_history.append([0, 0])

                # Finger gesture classification

                finger\_gesture\_id = 0

                point\_history\_len = len(pre\_processed\_point\_history\_list)

                if point\_history\_len == (history\_length \* 2):

                    finger\_gesture\_id = point\_history\_classifier(

                        pre\_processed\_point\_history\_list)

                # Calculates the gesture IDs in the latest detection

                finger\_gesture\_history.append(finger\_gesture\_id)

                most\_common\_fg\_id = Counter(

                    finger\_gesture\_history).most\_common()

                # Drawing part

                debug\_image = draw\_bounding\_rect(use\_brect, debug\_image, brect)

                debug\_image = draw\_landmarks(debug\_image, landmark\_list)

                debug\_image = draw\_info\_text(

                    debug\_image,

                    brect,

                    handedness,

                    keypoint\_classifier\_labels[hand\_sign\_id],

                    point\_history\_classifier\_labels[most\_common\_fg\_id[0][0]],

                )

        else:

            point\_history.append([0, 0])

        debug\_image = draw\_point\_history(debug\_image, point\_history)

        debug\_image = draw\_info(debug\_image, fps, mode, number)

        # Screen reflection #############################################################

        cv.imshow('Hand Gesture Recognition', debug\_image)

    cap.release()

    cv.destroyAllWindows()

def select\_mode(key, mode):

    number = -1

    if 48 <= key <= 57:  # 0 ~ 9

        number = key - 48

    if key == 110:  # n

        mode = 0

    if key == 107:  # k

        mode = 1

    if key == 104:  # h

        mode = 2

    return number, mode

def calc\_bounding\_rect(image, landmarks):

    image\_width, image\_height = image.shape[1], image.shape[0]

    landmark\_array = np.empty((0, 2), int)

    for \_, landmark in enumerate(landmarks.landmark):

        landmark\_x = min(int(landmark.x \* image\_width), image\_width - 1)

        landmark\_y = min(int(landmark.y \* image\_height), image\_height - 1)

        landmark\_point = [np.array((landmark\_x, landmark\_y))]

        landmark\_array = np.append(landmark\_array, landmark\_point, axis=0)

    x, y, w, h = cv.boundingRect(landmark\_array)

    return [x, y, x + w, y + h]

def calc\_landmark\_list(image, landmarks):

    image\_width, image\_height = image.shape[1], image.shape[0]

    landmark\_point = []

    # Keypoint

    for \_, landmark in enumerate(landmarks.landmark):

        landmark\_x = min(int(landmark.x \* image\_width), image\_width - 1)

        landmark\_y = min(int(landmark.y \* image\_height), image\_height - 1)

        # landmark\_z = landmark.z

        landmark\_point.append([landmark\_x, landmark\_y])

    return landmark\_point

def pre\_process\_landmark(landmark\_list):

    temp\_landmark\_list = copy.deepcopy(landmark\_list)

    # Convert to relative coordinates

    base\_x, base\_y = 0, 0

    for index, landmark\_point in enumerate(temp\_landmark\_list):

        if index == 0:

            base\_x, base\_y = landmark\_point[0], landmark\_point[1]

        temp\_landmark\_list[index][0] = temp\_landmark\_list[index][0] - base\_x

        temp\_landmark\_list[index][1] = temp\_landmark\_list[index][1] - base\_y

    # Convert to a one-dimensional list

    temp\_landmark\_list = list(

        itertools.chain.from\_iterable(temp\_landmark\_list))

    # Normalization

    max\_value = max(list(map(abs, temp\_landmark\_list)))

    def normalize\_(n):

        return n / max\_value

    temp\_landmark\_list = list(map(normalize\_, temp\_landmark\_list))

    return temp\_landmark\_list

def pre\_process\_point\_history(image, point\_history):

    image\_width, image\_height = image.shape[1], image.shape[0]

    temp\_point\_history = copy.deepcopy(point\_history)

    # Convert to relative coordinates

    base\_x, base\_y = 0, 0

    for index, point in enumerate(temp\_point\_history):

        if index == 0:

            base\_x, base\_y = point[0], point[1]

        temp\_point\_history[index][0] = (temp\_point\_history[index][0] -

                                        base\_x) / image\_width

        temp\_point\_history[index][1] = (temp\_point\_history[index][1] -

                                        base\_y) / image\_height

    # Convert to a one-dimensional list

    temp\_point\_history = list(

        itertools.chain.from\_iterable(temp\_point\_history))

    return temp\_point\_history

def logging\_csv(number, mode, landmark\_list, point\_history\_list):

    if mode == 0:

        pass

    if mode == 1 and (0 <= number <= 9):

        csv\_path = 'model/keypoint\_classifier/keypoint.csv'

        with open(csv\_path, 'a', newline="") as f:

            writer = csv.writer(f)

            writer.writerow([number, \*landmark\_list])

    if mode == 2 and (0 <= number <= 9):

        csv\_path = 'model/point\_history\_classifier/point\_history.csv'

        with open(csv\_path, 'a', newline="") as f:

            writer = csv.writer(f)

            writer.writerow([number, \*point\_history\_list])

    return

def draw\_landmarks(image, landmark\_point):

    if len(landmark\_point) > 0:

        # Thumb

        cv.line(image, tuple(landmark\_point[2]), tuple(landmark\_point[3]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[2]), tuple(landmark\_point[3]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[3]), tuple(landmark\_point[4]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[3]), tuple(landmark\_point[4]),

                (255, 255, 255), 2)

        # Index finger

        cv.line(image, tuple(landmark\_point[5]), tuple(landmark\_point[6]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[5]), tuple(landmark\_point[6]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[6]), tuple(landmark\_point[7]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[6]), tuple(landmark\_point[7]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[7]), tuple(landmark\_point[8]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[7]), tuple(landmark\_point[8]),

                (255, 255, 255), 2)

        # Middle finger

        cv.line(image, tuple(landmark\_point[9]), tuple(landmark\_point[10]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[9]), tuple(landmark\_point[10]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[10]), tuple(landmark\_point[11]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[10]), tuple(landmark\_point[11]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[11]), tuple(landmark\_point[12]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[11]), tuple(landmark\_point[12]),

                (255, 255, 255), 2)

        # Ring finger

        cv.line(image, tuple(landmark\_point[13]), tuple(landmark\_point[14]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[13]), tuple(landmark\_point[14]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[14]), tuple(landmark\_point[15]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[14]), tuple(landmark\_point[15]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[15]), tuple(landmark\_point[16]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[15]), tuple(landmark\_point[16]),

                (255, 255, 255), 2)

        # Little finger

        cv.line(image, tuple(landmark\_point[17]), tuple(landmark\_point[18]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[17]), tuple(landmark\_point[18]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[18]), tuple(landmark\_point[19]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[18]), tuple(landmark\_point[19]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[19]), tuple(landmark\_point[20]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[19]), tuple(landmark\_point[20]),

                (255, 255, 255), 2)

        # Palm

        cv.line(image, tuple(landmark\_point[0]), tuple(landmark\_point[1]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[0]), tuple(landmark\_point[1]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[1]), tuple(landmark\_point[2]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[1]), tuple(landmark\_point[2]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[2]), tuple(landmark\_point[5]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[2]), tuple(landmark\_point[5]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[5]), tuple(landmark\_point[9]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[5]), tuple(landmark\_point[9]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[9]), tuple(landmark\_point[13]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[9]), tuple(landmark\_point[13]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[13]), tuple(landmark\_point[17]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[13]), tuple(landmark\_point[17]),

                (255, 255, 255), 2)

        cv.line(image, tuple(landmark\_point[17]), tuple(landmark\_point[0]),

                (0, 0, 0), 6)

        cv.line(image, tuple(landmark\_point[17]), tuple(landmark\_point[0]),

                (255, 255, 255), 2)

    # Key Points

    for index, landmark in enumerate(landmark\_point):

        if index == 0:  # æ‰‹é¦–1

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 1:  # æ‰‹é¦–2

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 2:  # è¦ªæŒ‡ï¼šä»˜ã‘æ ¹

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 3:  # è¦ªæŒ‡ï¼šç¬¬1é–¢ç¯€

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 4:  # è¦ªæŒ‡ï¼šæŒ‡å…ˆ

            cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)

        if index == 5:  # äººå·®æŒ‡ï¼šä»˜ã‘æ ¹

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 6:  # äººå·®æŒ‡ï¼šç¬¬2é–¢ç¯€

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 7:  # äººå·®æŒ‡ï¼šç¬¬1é–¢ç¯€

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 8:  # äººå·®æŒ‡ï¼šæŒ‡å…ˆ

            cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)

        if index == 9:  # ä¸­æŒ‡ï¼šä»˜ã‘æ ¹

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 10:  # ä¸­æŒ‡ï¼šç¬¬2é–¢ç¯€

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 11:  # ä¸­æŒ‡ï¼šç¬¬1é–¢ç¯€

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 12:  # ä¸­æŒ‡ï¼šæŒ‡å…ˆ

            cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)

        if index == 13:  # è–¬æŒ‡ï¼šä»˜ã‘æ ¹

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 14:  # è–¬æŒ‡ï¼šç¬¬2é–¢ç¯€

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 15:  # è–¬æŒ‡ï¼šç¬¬1é–¢ç¯€

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 16:  # è–¬æŒ‡ï¼šæŒ‡å…ˆ

            cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)

        if index == 17:  # å°æŒ‡ï¼šä»˜ã‘æ ¹

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 18:  # å°æŒ‡ï¼šç¬¬2é–¢ç¯€

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 19:  # å°æŒ‡ï¼šç¬¬1é–¢ç¯€

            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)

        if index == 20:  # å°æŒ‡ï¼šæŒ‡å…ˆ

            cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),

                      -1)

            cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)

    return image

def draw\_bounding\_rect(use\_brect, image, brect):

    if use\_brect:

        # Outer rectangle

        cv.rectangle(image, (brect[0], brect[1]), (brect[2], brect[3]),

                     (0, 0, 0), 1)

    return image

def draw\_info\_text(image, brect, handedness, hand\_sign\_text,

                   finger\_gesture\_text):

    cv.rectangle(image, (brect[0], brect[1]), (brect[2], brect[1] - 22),

                 (0, 0, 0), -1)

    info\_text = handedness.classification[0].label[0:]

    if hand\_sign\_text != "":

        info\_text = info\_text + ':' + hand\_sign\_text

    cv.putText(image, info\_text, (brect[0] + 5, brect[1] - 4),

               cv.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 255, 255), 1, cv.LINE\_AA)

    if finger\_gesture\_text != "":

        cv.putText(image, "Finger Gesture:" + finger\_gesture\_text, (10, 60),

                   cv.FONT\_HERSHEY\_SIMPLEX, 1.0, (0, 0, 0), 4, cv.LINE\_AA)

        cv.putText(image, "Finger Gesture:" + finger\_gesture\_text, (10, 60),

                   cv.FONT\_HERSHEY\_SIMPLEX, 1.0, (255, 255, 255), 2,

                   cv.LINE\_AA)

    return image

def draw\_point\_history(image, point\_history):

    for index, point in enumerate(point\_history):

        if point[0] != 0 and point[1] != 0:

            cv.circle(image, (point[0], point[1]), 1 + int(index / 2),

                      (152, 251, 152), 2)

    return image

def draw\_info(image, fps, mode, number):

    cv.putText(image, "FPS:" + str(fps), (10, 30), cv.FONT\_HERSHEY\_SIMPLEX,

               1.0, (0, 0, 0), 4, cv.LINE\_AA)

    cv.putText(image, "FPS:" + str(fps), (10, 30), cv.FONT\_HERSHEY\_SIMPLEX,

               1.0, (255, 255, 255), 2, cv.LINE\_AA)

    mode\_string = ['Logging Key Point', 'Logging Point History']

    if 1 <= mode <= 2:

        cv.putText(image, "MODE:" + mode\_string[mode - 1], (10, 90),

                   cv.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 255, 255), 1,

                   cv.LINE\_AA)

        if 0 <= number <= 9:

            cv.putText(image, "NUM:" + str(number), (10, 110),

                       cv.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 255, 255), 1,

                       cv.LINE\_AA)

    return image

if \_\_name\_\_ == '\_\_main\_\_':

    main()