```python
import os
import csv
import copy
import argparse
import itertools

import cv2 as cv
import numpy as np
import mediapipe as mp

from utils.cvfpscalc import CvFpsCalc
from model.keypoint_classifier.keypoint_classifier import KeyPointClassifier


datasetdir = "model/dataset/dataset 1"


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=2,
        min_detection_confidence=min_detection_confidence,
        min_tracking_confidence=min_tracking_confidence,
    )

    keypoint_classifier = KeyPointClassifier()

    # 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]

    # FPS Measurement ########################################################
    cvFpsCalc = CvFpsCalc(buffer_len=10)

    #  ########################################################################
    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 mode == 2:
            # Loading image while processing the dataset
            loading_img = cv.imread("./assets/om606.png", cv.IMREAD_COLOR)

            cv.putText(
                loading_img,
                "Loading...",
```

```
                (20, 50),
                cv.FONT_HERSHEY_SIMPLEX,
                1.0,
                (255, 255, 255),
                4,
                cv.LINE_AA,
            )

            cv.imshow("Hand Gesture Recognition", loading_img)

            key = cv.waitKey(1000)

            # Looping through each folder of the dataset
            imglabel = -1
            for imgclass in os.listdir(datasetdir):
                imglabel += 1
                numofimgs = 0
                for img in os.listdir(os.path.join(datasetdir, imgclass)):
                    numofimgs += 1
                    imgpath = os.path.join(datasetdir, imgclass, img)
                    try:
                        img = cv.imread(imgpath)
                        debug_img = copy.deepcopy(img)

                        for _ in [1, 2]:
                            img.flags.writeable = False
                            results = hands.process(img)
                            img.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_img, hand_landmarks
                                    )
                                    # Landmark calculation
                                    landmark_list = calc_landmark_list(
                                        debug_img, hand_landmarks
                                    )

                                    # Conversion to relative coordinates / normalized coordinates
                                    pre_processed_landmark_list = pre_process_landmark(
                                        landmark_list
                                    )

                                    # Write to the dataset file
                                    logging_csv(
                                        imglabel, mode, pre_processed_landmark_list
                                    )
                            img = cv.flip(img, 0)
                    except Exception as e:
                        print(f"Issue with image {imgpath}")

                print(f"Num of image of the class {imglabel} is : {numofimgs}")
            mode = 1
```

```
            print("End of job!")
            break
        else:
            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)

                    # Write to the dataset file
                    logging_csv(number, mode, pre_processed_landmark_list)

                    # Hand sign classification
                    hand_sign_id = keypoint_classifier(pre_processed_landmark_list)

                    # Finger gesture classification
                    finger_gesture_id = 0

                    # 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],
                    )

            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 65 <= key <= 90:  # A ~ B
        number = key - 65
    if key == 110:  # n (Inference Mode)
        mode = 0
    if key == 107:  # k (Capturing Landmark From Camera Mode)
        mode = 1
    if key == 100:  # d (Capturing Landmarks From Provided Dataset Mode)
        mode = 2
    return number, mode


def calc_bounding_rect(image, landmarks):
    image_width, image_height = image.shape[1], image.shape[0]

```

```python
    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 logging_csv(number, mode, landmark_list):
    if mode == 0:
        pass
```

```python
    if (mode == 1 or mode == 2) and (0 <= number <= 35):
        csv_path = "model/keypoint_classifier/keypoint.csv"
        with open(csv_path, "a", newline="") as f:
            writer = csv.writer(f)
            writer.writerow([number, *landmark_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:
            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:  # 手首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:  # 親指：付け根
            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:  # 親指：第1関節
            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:  # 親指：指先
            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:  # 人差指: 付け根
            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:  # 人差指: 第2関節
            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:  # 人差指: 第1関節
            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:  # 人差指: 指先
            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:  # 中指: 付け根
            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:  # 中指: 第2関節
            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:  # 中指: 第1関節
            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:  # 中指: 指先
            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:  # 薬指: 付け根
            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:  # 薬指: 第2関節
            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:  # 薬指: 第1関節
            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:  # 薬指: 指先
            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:  # 小指: 付け根
            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:  # 小指: 第2関節
            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:  # 小指: 第1関節
            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:  # 小指: 指先
            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


```

```python
def draw_info_text(image, brect, handedness, hand_sign_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,
    )

    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",
        "Capturing Landmarks From Provided Dataset Mode",
    ]
    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()

```