

```

1 import os
2 import csv
3 import copy
4 import argparse
5 import itertools
6
7 import cv2 as cv
8 import numpy as np
9 import mediapipe as mp
10
11 from utils.cvfpscalc import CvFpsCalc
12 from model.keypoint_classifier.keypoint_classifier import KeyPointClassifier
13
14
15 datasetdir = "model/dataset/dataset 1"
16
17
18 def get_args():
19     parser = argparse.ArgumentParser()
20
21     parser.add_argument("--device", type=int, default=0)
22     parser.add_argument("--width", help="cap width", type=int, default=960)
23     parser.add_argument("--height", help="cap height", type=int, default=540)
24
25     parser.add_argument("--use_static_image_mode", action="store_true")
26     parser.add_argument(
27         "--min_detection_confidence",
28         help="min_detection_confidence",
29         type=float,
30         default=0.7,
31     )
32     parser.add_argument(
33         "--min_tracking_confidence",
34         help="min_tracking_confidence",
35         type=int,
36         default=0.5,
37     )
38
39     args = parser.parse_args()
40
41     return args
42
43
44 def main():
45     # Argument parsing
46     #####
47     args = get_args()
48
49     cap_device = args.device
50     cap_width = args.width
51     cap_height = args.height
52
53     use_static_image_mode = args.use_static_image_mode
54     min_detection_confidence = args.min_detection_confidence
55     min_tracking_confidence = args.min_tracking_confidence
56
57     use_brect = True
58
59     # Camera preparation
60     #####

```

```

59     cap = cv.VideoCapture(cap_device)
60     cap.set(cv.CAP_PROP_FRAME_WIDTH, cap_width)
61     cap.set(cv.CAP_PROP_FRAME_HEIGHT, cap_height)
62
63     # Model load #####
64     mp_hands = mp.solutions.hands
65     hands = mp_hands.Hands(
66         static_image_mode=use_static_image_mode,
67         max_num_hands=2,
68         min_detection_confidence=min_detection_confidence,
69         min_tracking_confidence=min_tracking_confidence,
70     )
71
72     keypoint_classifier = KeyPointClassifier()
73
74     # Read labels #####
75     with open(
76         "model/keypoint_classifier/keypoint_classifier_label.csv", encoding="utf-8-
sig"
77     ) as f:
78         keypoint_classifier_labels = csv.reader(f)
79         keypoint_classifier_labels = [row[0] for row in keypoint_classifier_labels]
80
81     # FPS Measurement #####
82     cvFpsCalc = CvFpsCalc(buffer_len=10)
83
84     # #####
85     mode = 0
86
87     while True:
88         fps = cvFpsCalc.get()
89
90         # Process Key (ESC: end) #####
91         key = cv.waitKey(10)
92         if key == 27: # ESC
93             break
94         number, mode = select_mode(key, mode)
95
96         # Camera capture #####
97         ret, image = cap.read()
98         if not ret:
99             break
100         image = cv.flip(image, 1) # Mirror display
101         debug_image = copy.deepcopy(image)
102
103         # Detection implementation
104         #####
105         image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
106
107         image.flags.writeable = False
108         results = hands.process(image)
109         image.flags.writeable = True
110
111         if mode == 2:
112             # Loading image while processing the dataset
113             loading_img = cv.imread("./assets/om606.png", cv.IMREAD_COLOR)
114
115             cv.putText(
116                 loading_img,

```

```

117         (20, 50),
118         cv.FONT_HERSHEY_SIMPLEX,
119         1.0,
120         (255, 255, 255),
121         4,
122         cv.LINE_AA,
123     )
124
125     cv.imshow("Hand Gesture Recognition", loading_img)
126
127     key = cv.waitKey(1000)
128
129     # Looping through each folder of the dataset
130     imglabel = -1
131     for imgclass in os.listdir(datasetdir):
132         imglabel += 1
133         numofimgs = 0
134         for img in os.listdir(os.path.join(datasetdir, imgclass)):
135             numofimgs += 1
136             imgpath = os.path.join(datasetdir, imgclass, img)
137             try:
138                 img = cv.imread(imgpath)
139                 debug_img = copy.deepcopy(img)
140
141                 for _ in [1, 2]:
142                     img.flags.writeable = False
143                     results = hands.process(img)
144                     img.flags.writeable = True
145
146                     if results.multi_hand_landmarks is not None:
147                         for hand_landmarks, handedness in zip(
148                             results.multi_hand_landmarks,
149                             results.multi_handedness,
150                         ):
151                             # Bounding box calculation
152                             brect = calc_bounding_rect(
153                                 debug_img, hand_landmarks
154                             )
155                             # Landmark calculation
156                             landmark_list = calc_landmark_list(
157                                 debug_img, hand_landmarks
158                             )
159
160                             # Conversion to relative coordinates / normalized
161                             coordinates
162                             pre_processed_landmark_list =
163                             pre_process_landmark(
164                                 landmark_list
165                             )
166
167                             # Write to the dataset file
168                             logging_csv(
169                                 imglabel, mode, pre_processed_landmark_list
170                             )
171                             img = cv.flip(img, 0)
172                         except Exception as e:
173                             print(f"Issue with image {imgpath}")
174
175                             print(f"Num of image of the class {imglabel} is : {numofimgs}")
176                             mode = 1

```

```

175         print("End of job!")
176         break
177     else:
178         if results.multi_hand_landmarks is not None:
179             for hand_landmarks, handedness in zip(
180                 results.multi_hand_landmarks, results.multi_handedness
181             ):
182                 # Bounding box calculation
183                 brect = calc_bounding_rect(debug_image, hand_landmarks)
184                 # Landmark calculation
185                 landmark_list = calc_landmark_list(debug_image, hand_landmarks)
186
187                 # Conversion to relative coordinates / normalized coordinates
188                 pre_processed_landmark_list = pre_process_landmark(landmark_list)
189
190                 # Write to the dataset file
191                 logging_csv(number, mode, pre_processed_landmark_list)
192
193                 # Hand sign classification
194                 hand_sign_id = keypoint_classifier(pre_processed_landmark_list)
195
196                 # Finger gesture classification
197                 finger_gesture_id = 0
198
199                 # Drawing part
200                 debug_image = draw_bounding_rect(use_brect, debug_image, brect)
201                 debug_image = draw_landmarks(debug_image, landmark_list)
202                 debug_image = draw_info_text(
203                     debug_image,
204                     brect,
205                     handedness,
206                     keypoint_classifier_labels[hand_sign_id],
207                 )
208
209                 debug_image = draw_info(debug_image, fps, mode, number)
210
211                 # Screen reflection
212                 #####
213                 cv.imshow("Hand Gesture Recognition", debug_image)
214
215                 cap.release()
216                 cv.destroyAllWindows()
217
218 def select_mode(key, mode):
219     number = -1
220     if 65 <= key <= 90: # A ~ B
221         number = key - 65
222     if key == 110: # n (Inference Mode)
223         mode = 0
224     if key == 107: # k (Capturing Landmark From Camera Mode)
225         mode = 1
226     if key == 100: # d (Capturing Landmarks From Provided Dataset Mode)
227         mode = 2
228     return number, mode
229
230
231 def calc_bounding_rect(image, landmarks):
232     image_width, image_height = image.shape[1], image.shape[0]
233

```

```
234     landmark_array = np.empty((0, 2), int)
235
236     for _, landmark in enumerate(landmarks.landmark):
237         landmark_x = min(int(landmark.x * image_width), image_width - 1)
238         landmark_y = min(int(landmark.y * image_height), image_height - 1)
239
240         landmark_point = [np.array((landmark_x, landmark_y))]
241
242         landmark_array = np.append(landmark_array, landmark_point, axis=0)
243
244     x, y, w, h = cv.boundingRect(landmark_array)
245
246     return [x, y, x + w, y + h]
247
248
249 def calc_landmark_list(image, landmarks):
250     image_width, image_height = image.shape[1], image.shape[0]
251
252     landmark_point = []
253
254     # Keypoint
255     for _, landmark in enumerate(landmarks.landmark):
256         landmark_x = min(int(landmark.x * image_width), image_width - 1)
257         landmark_y = min(int(landmark.y * image_height), image_height - 1)
258         # landmark_z = landmark.z
259
260         landmark_point.append([landmark_x, landmark_y])
261
262     return landmark_point
263
264
265 def pre_process_landmark(landmark_list):
266     temp_landmark_list = copy.deepcopy(landmark_list)
267
268     # Convert to relative coordinates
269     base_x, base_y = 0, 0
270     for index, landmark_point in enumerate(temp_landmark_list):
271         if index == 0:
272             base_x, base_y = landmark_point[0], landmark_point[1]
273
274         temp_landmark_list[index][0] = temp_landmark_list[index][0] - base_x
275         temp_landmark_list[index][1] = temp_landmark_list[index][1] - base_y
276
277     # Convert to a one-dimensional list
278     temp_landmark_list = list(itertools.chain.from_iterable(temp_landmark_list))
279
280     # Normalization
281     max_value = max(list(map(abs, temp_landmark_list)))
282
283     def normalize_(n):
284         return n / max_value
285
286     temp_landmark_list = list(map(normalize_, temp_landmark_list))
287
288     return temp_landmark_list
289
290
291 def logging_csv(number, mode, landmark_list):
292     if mode == 0:
293         pass
```

```
294     if (mode == 1 or mode == 2) and (0 <= number <= 35):
295         csv_path = "model/keypoint_classifier/keypoint.csv"
296         with open(csv_path, "a", newline="") as f:
297             writer = csv.writer(f)
298             writer.writerow([number, *landmark_list])
299     return
300
301
302 def draw_landmarks(image, landmark_point):
303     if len(landmark_point) > 0:
304         # Thumb
305         cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[3]), (0, 0, 0),
306 6)
307         cv.line(
308             image,
309             tuple(landmark_point[2]),
310             tuple(landmark_point[3]),
311             (255, 255, 255),
312             2,
313         )
314         cv.line(image, tuple(landmark_point[3]), tuple(landmark_point[4]), (0, 0, 0),
315 6)
316         cv.line(
317             image,
318             tuple(landmark_point[3]),
319             tuple(landmark_point[4]),
320             (255, 255, 255),
321             2,
322         )
323         # Index finger
324         cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[6]), (0, 0, 0),
325 6)
326         cv.line(
327             image,
328             tuple(landmark_point[5]),
329             tuple(landmark_point[6]),
330             (255, 255, 255),
331             2,
332         )
333         cv.line(image, tuple(landmark_point[6]), tuple(landmark_point[7]), (0, 0, 0),
334 6)
335         cv.line(
336             image,
337             tuple(landmark_point[6]),
338             tuple(landmark_point[7]),
339             (255, 255, 255),
340             2,
341         )
342         cv.line(image, tuple(landmark_point[7]), tuple(landmark_point[8]), (0, 0, 0),
343 6)
344         cv.line(
345             image,
346             tuple(landmark_point[7]),
347             tuple(landmark_point[8]),
348             (255, 255, 255),
349             2,
350         )
351         # Middle finger
```

```
349 cv.line(
350     image, tuple(landmark_point[9]), tuple(landmark_point[10]), (0, 0, 0), 6
351 )
352 cv.line(
353     image,
354     tuple(landmark_point[9]),
355     tuple(landmark_point[10]),
356     (255, 255, 255),
357     2,
358 )
359 cv.line(
360     image, tuple(landmark_point[10]), tuple(landmark_point[11]), (0, 0, 0), 6
361 )
362 cv.line(
363     image,
364     tuple(landmark_point[10]),
365     tuple(landmark_point[11]),
366     (255, 255, 255),
367     2,
368 )
369 cv.line(
370     image, tuple(landmark_point[11]), tuple(landmark_point[12]), (0, 0, 0), 6
371 )
372 cv.line(
373     image,
374     tuple(landmark_point[11]),
375     tuple(landmark_point[12]),
376     (255, 255, 255),
377     2,
378 )
379
380 # Ring finger
381 cv.line(
382     image, tuple(landmark_point[13]), tuple(landmark_point[14]), (0, 0, 0), 6
383 )
384 cv.line(
385     image,
386     tuple(landmark_point[13]),
387     tuple(landmark_point[14]),
388     (255, 255, 255),
389     2,
390 )
391 cv.line(
392     image, tuple(landmark_point[14]), tuple(landmark_point[15]), (0, 0, 0), 6
393 )
394 cv.line(
395     image,
396     tuple(landmark_point[14]),
397     tuple(landmark_point[15]),
398     (255, 255, 255),
399     2,
400 )
401 cv.line(
402     image, tuple(landmark_point[15]), tuple(landmark_point[16]), (0, 0, 0), 6
403 )
404 cv.line(
405     image,
406     tuple(landmark_point[15]),
407     tuple(landmark_point[16]),
408     (255, 255, 255),
```

```
409         2,
410     )
411
412     # Little finger
413     cv.line(
414         image, tuple(landmark_point[17]), tuple(landmark_point[18]), (0, 0, 0), 6
415     )
416     cv.line(
417         image,
418         tuple(landmark_point[17]),
419         tuple(landmark_point[18]),
420         (255, 255, 255),
421         2,
422     )
423     cv.line(
424         image, tuple(landmark_point[18]), tuple(landmark_point[19]), (0, 0, 0), 6
425     )
426     cv.line(
427         image,
428         tuple(landmark_point[18]),
429         tuple(landmark_point[19]),
430         (255, 255, 255),
431         2,
432     )
433     cv.line(
434         image, tuple(landmark_point[19]), tuple(landmark_point[20]), (0, 0, 0), 6
435     )
436     cv.line(
437         image,
438         tuple(landmark_point[19]),
439         tuple(landmark_point[20]),
440         (255, 255, 255),
441         2,
442     )
443
444     # Palm
445     cv.line(image, tuple(landmark_point[0]), tuple(landmark_point[1]), (0, 0, 0),
6)
446     cv.line(
447         image,
448         tuple(landmark_point[0]),
449         tuple(landmark_point[1]),
450         (255, 255, 255),
451         2,
452     )
453     cv.line(image, tuple(landmark_point[1]), tuple(landmark_point[2]), (0, 0, 0),
6)
454     cv.line(
455         image,
456         tuple(landmark_point[1]),
457         tuple(landmark_point[2]),
458         (255, 255, 255),
459         2,
460     )
461     cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[5]), (0, 0, 0),
6)
462     cv.line(
463         image,
464         tuple(landmark_point[2]),
465         tuple(landmark_point[5]),
```



```

466         (255, 255, 255),
467         2,
468     )
469     cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[9]), (0, 0, 0),
6)
470     cv.line(
471         image,
472         tuple(landmark_point[5]),
473         tuple(landmark_point[9]),
474         (255, 255, 255),
475         2,
476     )
477     cv.line(
478         image, tuple(landmark_point[9]), tuple(landmark_point[13]), (0, 0, 0), 6
479     )
480     cv.line(
481         image,
482         tuple(landmark_point[9]),
483         tuple(landmark_point[13]),
484         (255, 255, 255),
485         2,
486     )
487     cv.line(
488         image, tuple(landmark_point[13]), tuple(landmark_point[17]), (0, 0, 0), 6
489     )
490     cv.line(
491         image,
492         tuple(landmark_point[13]),
493         tuple(landmark_point[17]),
494         (255, 255, 255),
495         2,
496     )
497     cv.line(
498         image, tuple(landmark_point[17]), tuple(landmark_point[0]), (0, 0, 0), 6
499     )
500     cv.line(
501         image,
502         tuple(landmark_point[17]),
503         tuple(landmark_point[0]),
504         (255, 255, 255),
505         2,
506     )
507
508     # Key Points
509     for index, landmark in enumerate(landmark_point):
510         if index == 0:
511             cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
512             cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
513         if index == 1: # 手首2
514             cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
515             cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
516         if index == 2: # 親指: 付け根
517             cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
518             cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
519         if index == 3: # 親指: 第1関節
520             cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
521             cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
522         if index == 4: # 親指: 指先
523             cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255), -1)
524             cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)

```

```

525     if index == 5: # 人差指: 付け根
526         cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
527         cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
528     if index == 6: # 人差指: 第2関節
529         cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
530         cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
531     if index == 7: # 人差指: 第1関節
532         cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
533         cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
534     if index == 8: # 人差指: 指先
535         cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255), -1)
536         cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
537     if index == 9: # 中指: 付け根
538         cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
539         cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
540     if index == 10: # 中指: 第2関節
541         cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
542         cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
543     if index == 11: # 中指: 第1関節
544         cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
545         cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
546     if index == 12: # 中指: 指先
547         cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255), -1)
548         cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
549     if index == 13: # 薬指: 付け根
550         cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
551         cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
552     if index == 14: # 薬指: 第2関節
553         cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
554         cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
555     if index == 15: # 薬指: 第1関節
556         cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
557         cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
558     if index == 16: # 薬指: 指先
559         cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255), -1)
560         cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
561     if index == 17: # 小指: 付け根
562         cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
563         cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
564     if index == 18: # 小指: 第2関節
565         cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
566         cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
567     if index == 19: # 小指: 第1関節
568         cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255), -1)
569         cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
570     if index == 20: # 小指: 指先
571         cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255), -1)
572         cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
573
574     return image
575
576
577 def draw_bounding_rect(use_brect, image, brect):
578     if use_brect:
579         # Outer rectangle
580         cv.rectangle(image, (brect[0], brect[1]), (brect[2], brect[3]), (0, 0, 0), 1)
581
582     return image
583
584

```

```
585 def draw_info_text(image, brect, handedness, hand_sign_text):
586     cv.rectangle(image, (brect[0], brect[1]), (brect[2], brect[1] - 22), (0, 0, 0),
587 -1)
588     info_text = handedness.classification[0].label[0:]
589     if hand_sign_text != "":
590         info_text = info_text + ":" + hand_sign_text
591     cv.putText(
592         image,
593         info_text,
594         (brect[0] + 5, brect[1] - 4),
595         cv.FONT_HERSHEY_SIMPLEX,
596         0.6,
597         (255, 255, 255),
598         1,
599         cv.LINE_AA,
600     )
601
602     return image
603
604
605 def draw_info(image, fps, mode, number):
606     cv.putText(
607         image,
608         "FPS:" + str(fps),
609         (10, 30),
610         cv.FONT_HERSHEY_SIMPLEX,
611         1.0,
612         (0, 0, 0),
613         4,
614         cv.LINE_AA,
615     )
616     cv.putText(
617         image,
618         "FPS:" + str(fps),
619         (10, 30),
620         cv.FONT_HERSHEY_SIMPLEX,
621         1.0,
622         (255, 255, 255),
623         2,
624         cv.LINE_AA,
625     )
626
627     mode_string = [
628         "Logging Key Point",
629         "Capturing Landmarks From Provided Dataset Mode",
630     ]
631     if 1 <= mode <= 2:
632         cv.putText(
633             image,
634             "MODE:" + mode_string[mode - 1],
635             (10, 90),
636             cv.FONT_HERSHEY_SIMPLEX,
637             0.6,
638             (255, 255, 255),
639             1,
640             cv.LINE_AA,
641         )
642     if 0 <= number <= 9:
643         cv.putText(
```

```
644         image,  
645         "NUM:" + str(number),  
646         (10, 110),  
647         cv.FONT_HERSHEY_SIMPLEX,  
648         0.6,  
649         (255, 255, 255),  
650         1,  
651         cv.LINE_AA,  
652     )  
653     return image  
654  
655  
656 if __name__ == "__main__":  
657     main()  
658
```