Appendix

Hand Tracking Module File:

```
import cv2
 import mediapipe as mp
 import math


 class handDetector:


 def __init__(self, mode=False, maxHands=2, detectionCon=0.5, minTrackCon=0.5):
 """
 :param mode: In static mode, detection is done on each image: slower
 :param maxHands: Maximum number of hands to detect
 :param detectionCon: Minimum Detection Confidence Threshold
 :param minTrackCon: Minimum Tracking Confidence Threshold
 """
 self.mode = mode
 self.maxHands = maxHands
 self.detectionCon = detectionCon
 self.minTrackCon = minTrackCon

 self.mpHands = mp.solutions.hands
 self.hands = self.mpHands.Hands(static_image_mode=self.mode,
max_num_hands=self.maxHands,
 min_detection_confidence=self.detectionCon, min_tracking_confidence=self.minTrackCon)
 self.mpDraw = mp.solutions.drawing_utils
 self.tipIds = [4, 8, 12, 16, 20]
 self.fingers = []
 self.lmList = []

 def findHands(self, img, draw=True):



"""
 Finds hands in a BGR image.
 :param img: Image to find the hands in.
 :param draw: Flag to draw the output on the image.
 :return: Image with or without drawings
 """
 imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
 self.results = self.hands.process(imgRGB)

 if self.results.multi_hand_landmarks:
 for handLms in self.results.multi_hand_landmarks:
 if draw:
 self.mpDraw.draw_landmarks(img, handLms,
 self.mpHands.HAND_CONNECTIONS)

 return img
 def findPosition(self, img, handNo=0, draw=True):
 """
 :param img: Image to find the hand's position.

:param handNo: Index number of hands to find position
 :param draw: Flag to draw the output on the image.
 :return: list of fingers and bbox.

"""
 xList = []
```

```
yList = []
bbox = []
self.lmList = []
if self.results.multi_hand_landmarks:
myHand = self.results.multi_hand_landmarks[handNo]
for id, lmin enumerate(myHand.landmark):
h, w, c = img. shape
px, py = int(lm.x * w), int(lm.y * h)
xList.append(px)
yList.append(py)
self.lmList.append([px, py])



if draw:
cv2.circle(img, (px, py), 5, (0, 255, 0), cv2.FILLED)
xmin, xmax = min(xList), max(xList)
ymin, maxi = min(yList), max(yList)
boxW, boxH = xmax - xmin, ymax - ymin
bbox = xmin, ymin, boxW, boxH
cx, cy = bbox[0] + (bbox[2] // 2), \
          bbox[1] + (bbox[3] // 2)

return self.lmList, bbox
def findDistance(self, p1, p2, img, draw=True):

"""
Find the distance between two landmarks based on their
index numbers.
:param p1: Point1
:param p2: Point2
```

```
:param img: Image to draw on.
:param draw: Flag to draw the output on the image.

:return: Distance between the points,
Image with output drawn,
Line information
 """
if self.results.multi_hand_landmarks:
x1, y1 = self.lmList[p1][0], self.lmList[p1][1]
x2, y2 = self.lmList[p2][0], self.lmList[p2][1]
cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
if draw:
            cv2.circle(img, (x1, y1), 5, (0, 255, 0), cv2.FILLED)
            cv2.circle(img, (x2, y2), 5, (0, 255, 0), cv2.FILLED)
            cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 3)
            cv2.circle(img, (cx, cy), 5, (0, 255, 0), cv2.FILLED)

length = math.hypot(x2 - x1, y2 - y1)
return length, img, [cx, cy]
```

Finger Counting File:

```
import cv2
import cvlearn.HandTrackingModule as htm

cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)

# print(len(overlayList))

detector = htm.handDetector(detectionCon=0.75)
```

```
tipIds = [4, 8, 12, 16, 20]

class FingerCounter:
"""A class to count Fingers"""
def __init__(self):
self.hi = ""

def drawCountedFingers(self, img, lmList, bbox):
"""
:param img: Image of hand to count fingers in.
:param lmList: list returned by find position in HandTrackingModule
:param bbox: bbox returned by find position in HandTrackingModule
"""
fingers = []

if lmList:
if lmList[5][0] >lmList[17][0]:

if lmList[tipIds[0]][0] >lmList[tipIds[0] - 1][0]:
fingers.append(1)
else:
fingers.append(0)
for id in range(1, 5):
if lmList[tipIds[id]][1] <lmList[tipIds[id] - 1][1]:
fingers.append(1)
else:
fingers.append(0)
img = cv2.putText(img, f"{fingers.count(1)}", (500, 100), cv2.FONT_HERSHEY_PLAIN, 5,
(255, 255, 0), 5)

if lmList[5][0] <lmList[17][0]:

if lmList[tipIds[0]][0] <lmList[tipIds[0] - 1][0]:
fingers.append(1)
else:
fingers.append(0)
for id in range(1, 5):
if lmList[tipIds[id]][1] <lmList[tipIds[id] - 1][1]:
fingers.append(1)
else:
fingers.append(0)
img = cv2.putText(img, f"{fingers.count(1)}", (10, 100), cv2.FONT_HERSHEY_PLAIN, 5,
(255, 255, 0), 5)
cv2.rectangle(img, (bbox[0] - 20, bbox[1] - 20),
(bbox[0] + bbox[2] + 20, bbox[1] + bbox[3] + 20),
(255, 0, 255), 2)

def countFingers(self, lmList):
"""
:param lmList: list returned by find position in HandTrackingModule
:return: List of fingers up or down
"""
fingers = []

if lmList:
if lmList[5][0] >lmList[17][0]:

if lmList[tipIds[0]][0] >lmList[tipIds[0] - 1][0]:
fingers.append(1)
else:
fingers.append(0)
for id in range(1, 5):
if lmList[tipIds[id]][1] <lmList[tipIds[id] - 1][1]:
fingers.append(1)
else:
fingers.append(0)
if lmList[5][0] <lmList[17][0]:

if lmList[tipIds[0]][0] <lmList[tipIds[0] - 1][0]:
fingers.append(1)



else:
fingers.append(0)
for id in range(1, 5):
if lmList[tipIds[id]][1] <lmList[tipIds[id] - 1][1]:
fingers.append(1)
else:
fingers.append(0)
totalFingers = fingers.count(1)
return totalFingers
```

Recognition Algorithm Using Finger Counting Main File:

```
import cv2
 import cvlearn.HandTrackingModule as htm
 import cvlearn.FingerCounter as counter
 cap = cv2.VideoCapture(0)
 detector = htm.handDetector()
 fingerCounter = counter.FingerCounter()



while True:
 ret, frame = cap.read()
 detector.findHands(frame)
 lmList, bbox = detector.findPosition(frame)
 if lmList != 0:
 fingerCounter.drawCountedFingers(frame, lmList, bbox)
 cv2.imshow("result", frame)
cv2.waitKey(1)
```

Binarization Algorithm Using Finger Counting Main File:

```
import cv2
 import mediapipe as mp

 cap = cv2.VideoCapture(0)
 mpHands = mp.solutions.hands
 hands = mpHands.Hands()
 mpDraw = mp.solutions.drawing_utils
 fingerCoordinates = [(8, 6), (12, 10), (16, 14), (20, 18)]
```

```
thumbCoordinate = (4,2)

while True:
success, img = cap.read()
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
results = hands.process(imgRGB)
multiLandMarks = results.multi_hand_landmarks

if multiLandMarks:
handPoints = []
for handLmsinmultiLandMarks:
mpDraw.draw_landmarks(img, handLms, mpHands.HAND_CONNECTIONS)

for idx, lmin enumerate(handLms.landmark):
# print(idx,lm)
h, w, c = img.shape
cx, cy = int(lm.x * w), int(lm.y * h)
handPoints.append((cx, cy))

for point in handPoints:
cv2.circle(img, point, 5, (0, 0, 255), cv2.FILLED)

upCount = 0
for coordinate in fingerCoordinates:
if handPoints[coordinate[0]][1] <handPoints[coordinate[1]][1]:
upCount += 1
if handPoints[thumbCoordinate[0]][0] >handPoints[thumbCoordinate[1]][0]:
upCount += 1

cv2.putText(img, str(upCount), (150,150), cv2.FONT_HERSHEY_PLAIN, 10, (255,0,0), 10)

cv2.imshow("result", img)
cv2.waitKey(1)
```

Scale-invariant Feature Transform (SHIFT) Algorithm Using Finger Counting Main File:

```
import cv2
 import mediapipe as mp
```

```
cap = cv2.VideoCapture(0)
mpHands = mp.solutions.hands
hands = mpHands.Hands()
mpDraw = mp.solutions.drawing_utils
fingerCoordinates = [(8, 6), (12, 10), (16, 14), (20, 18)]
thumbCoordinate = (4, 2)

while True:
        success, img = cap.read()
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        results = hands.process(imgRGB)
multiLandMarks = results.multi_hand_landmarks

if multiLandMarks:
handPoints = []
for handLmsinmultiLandMarks:
mpDraw.draw_landmarks(img, handLms, mpHands.HAND_CONNECTIONS)

for idx, lmin enumerate(handLms.landmark):
# print(idx,lm)
h, w, c = img.shape
cx, cy = int(lm.x * w), int(lm.y * h)
handPoints.append((cx, cy))

for point in handPoints:
cv2.circle(img, point, 3, (100, 0, 100), cv2.FILLED)

upCount = 0
for coordinate in fingerCoordinates:
if handPoints[coordinate[0]][1] <handPoints[coordinate[1]][1]:
upCount += 1
if handPoints[thumbCoordinate[0]][0] >handPoints[thumbCoordinate[1]][0]:
upCount += 1

cv2.putText(img, str(upCount), (20,100), cv2.FONT_HERSHEY_PLAIN, 8, (255,0,200), 8)
cv2.imshow("result", img)
cv2.waitKey(1)
```