### 09/18/2024

Follow through on the gaze detection model:

By downsampling (5fps), the model couldn’t process the passing of time as well as it could have. That being said, 10 fps did far better than I had imagined. I did not parse the entire video but I did for about 15 or so seconds of it to cut down on images (120).

The model actually catches the moments of distraction much better and displays it rather well with the distracted warning popping up more frequently than it had in the 5FPS processed videos.

The changes to the original codes are mundane, but I changed the main.py file, added a create\_original\_video.py file, process\_images\_to\_video.py file, and combine\_videos.py file.

They all would reference the folder I have the 10 fps images to make them into a video (without processing), process the images and make them into a video, then taking the processed video and unprocessed video and stitching them together side by side.

| main.py for 10 FPS |
| --- |
| import os  import time  import pprint  import cv2  import mediapipe as mp  import numpy as np  from attention\_scorer import AttentionScorer as AttScorer  from eye\_detector import EyeDetector as EyeDet  from parser import get\_args  from pose\_estimation import HeadPoseEstimator as HeadPoseEst  from utils import get\_landmarks, load\_camera\_parameters  def draw\_text\_with\_background(image, text, position, font\_scale, thickness, color, bg\_color, padding=10):  # Set font  font = cv2.FONT\_HERSHEY\_PLAIN  # Get the size of the text  (text\_width, text\_height), \_ = cv2.getTextSize(text, font, font\_scale, thickness)    # Set coordinates for the background rectangle  top\_left = (position[0] - padding, position[1] - padding)  bottom\_right = (position[0] + text\_width + padding, position[1] + text\_height + padding)  # Draw the rectangle  cv2.rectangle(image, top\_left, bottom\_right, bg\_color, -1)  # Draw the text over the rectangle  cv2.putText(image, text, position, font, font\_scale, color, thickness, cv2.LINE\_AA)  def main():  args = get\_args()  # Set the input and output directories  args.input\_dir = r"C:\Users\mc2di\project\_1\Driver-State-Detection\output\_images\Ramesh\_Video\_to\_Images\_10\_FPS"  args.output\_dir = r"C:\Users\mc2di\project\_1\Driver-State-Detection\processed\_images\_10FPS"  # Create the output directory if it doesn't exist  if not os.path.exists(args.output\_dir):  os.makedirs(args.output\_dir)  if not cv2.useOptimized():  try:  cv2.setUseOptimized(True) # Set OpenCV optimization to True  except:  print("OpenCV optimization could not be set to True, the script may be slower than expected")  if args.camera\_params:  camera\_matrix, dist\_coeffs = load\_camera\_parameters(args.camera\_params)  else:  camera\_matrix, dist\_coeffs = None, None  if args.verbose:  print("Arguments and Parameters used:\n")  pprint.pp(vars(args), indent=4)  print("\nCamera Matrix:")  pprint.pp(camera\_matrix, indent=4)  print("\nDistortion Coefficients:")  pprint.pp(dist\_coeffs, indent=4)  print("\n")  Detector = mp.solutions.face\_mesh.FaceMesh(  static\_image\_mode=False,  min\_detection\_confidence=0.5,  min\_tracking\_confidence=0.5,  refine\_landmarks=True,  )  Eye\_det = EyeDet(show\_processing=args.show\_eye\_proc)  Head\_pose = HeadPoseEst(  show\_axis=args.show\_axis, camera\_matrix=camera\_matrix, dist\_coeffs=dist\_coeffs  )  prev\_time = time.perf\_counter()  # Initialize Scorer before the loop, so it's accessible  Scorer = AttScorer(  t\_now=prev\_time, # Initialize with the initial time  ear\_thresh=args.ear\_thresh,  gaze\_time\_thresh=args.gaze\_time\_thresh,  roll\_thresh=args.roll\_thresh,  pitch\_thresh=args.pitch\_thresh,  yaw\_thresh=args.yaw\_thresh,  ear\_time\_thresh=args.ear\_time\_thresh,  gaze\_thresh=args.gaze\_thresh,  pose\_time\_thresh=args.pose\_time\_thresh,  verbose=args.verbose,  )  # Read images from the specified directory  image\_files = sorted([f for f in os.listdir(args.input\_dir) if f.endswith(('.jpg', '.png'))])  if not image\_files:  print(f"No images found in directory: {args.input\_dir}")  return  for image\_file in image\_files:  image\_path = os.path.join(args.input\_dir, image\_file)  frame = cv2.imread(image\_path)  if frame is None:  print(f"Error loading image: {image\_path}")  continue  # Calculate the current time and the time delta to measure FPS  t\_now = time.perf\_counter()  delta\_time = t\_now - prev\_time  if delta\_time > 0:  fps = 1.0 / delta\_time # Calculate frames per second  prev\_time = t\_now  # Initialize the tick counter for processing time measurement  e1 = cv2.getTickCount()  gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  frame\_size = frame.shape[1], frame.shape[0]  gray = np.expand\_dims(gray, axis=2)  gray = np.concatenate([gray, gray, gray], axis=2)  lms = Detector.process(gray).multi\_face\_landmarks  if lms:  landmarks = get\_landmarks(lms)  Eye\_det.show\_eye\_keypoints(color\_frame=frame, landmarks=landmarks, frame\_size=frame\_size)  ear = Eye\_det.get\_EAR(frame=gray, landmarks=landmarks)  tired, perclos\_score = Scorer.get\_PERCLOS(t\_now, fps, ear)  gaze = Eye\_det.get\_Gaze\_Score(frame=gray, landmarks=landmarks, frame\_size=frame\_size)  frame\_det, roll, pitch, yaw = Head\_pose.get\_pose(frame=frame, landmarks=landmarks, frame\_size=frame\_size)  asleep, looking\_away, distracted = Scorer.eval\_scores(  t\_now=t\_now,  ear\_score=ear,  gaze\_score=gaze,  head\_roll=roll,  head\_pitch=pitch,  head\_yaw=yaw,  )  if frame\_det is not None:  frame = frame\_det  # Draw text with larger font and darker background  if ear is not None:  draw\_text\_with\_background(frame, f"EAR: {round(ear, 3)}", (10, 50), 2, 2, (255, 255, 255), (0, 0, 0))  if gaze is not None:  draw\_text\_with\_background(frame, f"Gaze Score: {round(gaze, 3)}", (10, 80), 2, 2, (255, 255, 255), (0, 0, 0))  draw\_text\_with\_background(frame, f"PERCLOS: {round(perclos\_score, 3)}", (10, 110), 2, 2, (255, 255, 255), (0, 0, 0))  if roll is not None:  draw\_text\_with\_background(frame, f"roll: {roll.round(1)[0]}", (450, 40), 1.5, 1, (255, 0, 255), (0, 0, 0))  if pitch is not None:  draw\_text\_with\_background(frame, f"pitch: {pitch.round(1)[0]}", (450, 70), 1.5, 1, (255, 0, 255), (0, 0, 0))  if yaw is not None:  draw\_text\_with\_background(frame, f"yaw: {yaw.round(1)[0]}", (450, 100), 1.5, 1, (255, 0, 255), (0, 0, 0))  if tired:  draw\_text\_with\_background(frame, "TIRED!", (10, 280), 1, 1, (0, 0, 255), (0, 0, 0))  if asleep:  draw\_text\_with\_background(frame, "ASLEEP!", (10, 300), 1, 1, (0, 0, 255), (0, 0, 0))  if looking\_away:  draw\_text\_with\_background(frame, "LOOKING AWAY!", (10, 320), 1, 1, (0, 0, 255), (0, 0, 0))  if distracted:  draw\_text\_with\_background(frame, "DISTRACTED!", (10, 340), 1, 1, (0, 0, 255), (0, 0, 0))  e2 = cv2.getTickCount()  proc\_time\_frame\_ms = ((e2 - e1) / cv2.getTickFrequency()) \* 1000  if args.show\_fps:  draw\_text\_with\_background(frame, f"FPS: {round(fps)}", (10, 400), 2, 2, (255, 0, 255), (0, 0, 0))  if args.show\_proc\_time:  draw\_text\_with\_background(frame, f"PROC. TIME FRAME: {round(proc\_time\_frame\_ms, 0)}ms", (10, 430), 2, 2, (255, 0, 255), (0, 0, 0))  # Save the processed image  output\_path = os.path.join(args.output\_dir, image\_file)  cv2.imwrite(output\_path, frame)  # Optionally display the processed frame  cv2.imshow("Processed Frame", frame)  if cv2.waitKey(20) & 0xFF == ord("q"):  break  cv2.destroyAllWindows()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

| Create\_original\_video.py for 10FPS |
| --- |
| import cv2  import os  # Path to original images folder and output video  input\_folder = r'C:\Users\mc2di\project\_1\Driver-State-Detection\output\_images\Ramesh\_Video\_to\_Images\_10\_FPS'  output\_video\_path\_original = r'C:\Users\mc2di\project\_1\Driver-State-Detection\original\_video2.mp4'  # Parameters  frame\_rate = 5 # 10 FPS as per the images' extraction rate  frame\_size = None # Will be determined from the first image  def load\_images\_from\_folder(folder):  if not os.path.exists(folder):  raise FileNotFoundError(f"Input folder not found: {folder}")    images = []  filenames = sorted([f for f in os.listdir(folder) if f.endswith(('.jpg', '.png'))])    for filename in filenames:  img = cv2.imread(os.path.join(folder, filename))  if img is not None:  images.append(img)    return images  # Print the path for debugging  print(f"Input folder: {input\_folder}")  # Load images  original\_images = load\_images\_from\_folder(input\_folder)  # Ensure images are loaded  if not original\_images:  raise ValueError("No images found in the input folder.")  # Determine frame size from the first image  height, width, \_ = original\_images[0].shape  frame\_size = (width, height)  # Create VideoWriter object  fourcc = cv2.VideoWriter\_fourcc(\*'XVID') # Use 'XVID' or 'MJPG' as needed  out = cv2.VideoWriter(output\_video\_path\_original, fourcc, frame\_rate, frame\_size)  if not out.isOpened():  raise ValueError("Error: Could not open video file for writing.")  # Write frames to video  for img in original\_images:  out.write(img)  # Release video writer  out.release()  print(f"Original video saved successfully at {output\_video\_path\_original}") |

| processed\_images\_to\_video.py |
| --- |
| import cv2  import os  def create\_video\_from\_images(image\_folder, video\_name, fps):  # Get a list of all image files in the output directory  image\_files = sorted([f for f in os.listdir(image\_folder) if f.endswith(('.png', '.jpg'))])  if not image\_files:  print(f"No images found in directory: {image\_folder}")  return  # Load the first image to get the frame dimensions  first\_image\_path = os.path.join(image\_folder, image\_files[0])  frame = cv2.imread(first\_image\_path)  height, width, layers = frame.shape  # Initialize the VideoWriter  fourcc = cv2.VideoWriter\_fourcc(\*'mp4v') # Use 'mp4v' codec for mp4 video  video = cv2.VideoWriter(video\_name, fourcc, fps, (width, height))  # Iterate through the sorted image files and add them to the video  for image\_file in image\_files:  image\_path = os.path.join(image\_folder, image\_file)  frame = cv2.imread(image\_path)  if frame is None:  print(f"Error loading image: {image\_path}")  continue  video.write(frame) # Write each frame to the video  # Release the VideoWriter  video.release()  print(f"Video saved as {video\_name}")  if \_\_name\_\_ == "\_\_main\_\_":  # Specify the directory containing the processed images  processed\_image\_dir = r"C:\Users\mc2di\project\_1\Driver-State-Detection\processed\_images\_10FPS"    # Specify the name and location of the output video  output\_video\_path = r"C:\Users\mc2di\project\_1\Driver-State-Detection\output\_video2.mp4"    # Specify the frame rate (adjust based on the original video)  fps = 10 # or the original FPS of the video  create\_video\_from\_images(processed\_image\_dir, output\_video\_path, fps) |

| combine\_videos.py |
| --- |
| import cv2  import os  def generate\_processed\_video(input\_dir, output\_video\_path, frame\_rate=30):  # Read image files  image\_files = sorted([f for f in os.listdir(input\_dir) if f.endswith(('.jpg', '.png'))])    if not image\_files:  raise ValueError("No images found in the directory.")    # Get dimensions from the first image  first\_image = cv2.imread(os.path.join(input\_dir, image\_files[0]))  height, width, \_ = first\_image.shape  # Create VideoWriter  fourcc = cv2.VideoWriter\_fourcc(\*'MJPG')  video\_writer = cv2.VideoWriter(output\_video\_path, fourcc, frame\_rate, (width, height))  # Write each image to the video  for image\_file in image\_files:  image\_path = os.path.join(input\_dir, image\_file)  image = cv2.imread(image\_path)  if image is None:  print(f"Error loading image: {image\_path}")  continue  video\_writer.write(image)  video\_writer.release()  print(f"Processed video saved successfully at {output\_video\_path}")  # Example usage  input\_images\_dir = r'C:\Users\mc2di\project\_1\Driver-State-Detection\processed\_images\_10FPS'  processed\_video\_path = r'C:\Users\mc2di\project\_1\Driver-State-Detection\processed\_video\_10FPS.mp4'  generate\_processed\_video(input\_images\_dir, processed\_video\_path)  import cv2  import numpy as np  import os  def combine\_videos(video\_path\_original, video\_path\_processed, output\_combined\_video\_path):  # Check if the files exist  if not os.path.isfile(video\_path\_original):  raise ValueError(f"Error: Original video file does not exist at {video\_path\_original}")  if not os.path.isfile(video\_path\_processed):  raise ValueError(f"Error: Processed video file does not exist at {video\_path\_processed}")  # Open the original and processed videos  cap\_original = cv2.VideoCapture(video\_path\_original)  cap\_processed = cv2.VideoCapture(video\_path\_processed)  if not cap\_original.isOpened():  raise ValueError(f"Error: Could not open original video file at {video\_path\_original}")  if not cap\_processed.isOpened():  raise ValueError(f"Error: Could not open processed video file at {video\_path\_processed}")  # Get video properties  frame\_width = int(cap\_original.get(cv2.CAP\_PROP\_FRAME\_WIDTH))  frame\_height = int(cap\_original.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))  fps = cap\_original.get(cv2.CAP\_PROP\_FPS)  # Create VideoWriter object for the combined video  fourcc = cv2.VideoWriter\_fourcc(\*'XVID') # Try 'MJPG' if 'XVID' does not work  out = cv2.VideoWriter(output\_combined\_video\_path, fourcc, fps, (frame\_width \* 2, frame\_height))  if not out.isOpened():  raise ValueError(f"Error: Could not open combined video file for writing at {output\_combined\_video\_path}")  while cap\_original.isOpened() and cap\_processed.isOpened():  ret\_original, frame\_original = cap\_original.read()  ret\_processed, frame\_processed = cap\_processed.read()    if not ret\_original or not ret\_processed:  print("Error reading frames from one or both videos.")  break    # Concatenate images horizontally  combined\_frame = np.hstack((frame\_original, frame\_processed))    # Write combined frame to video  out.write(combined\_frame)  # Release video objects  cap\_original.release()  cap\_processed.release()  out.release()  print(f"Combined video saved successfully at {output\_combined\_video\_path}")  # Example usage  original\_video\_path = r'C:\Users\mc2di\project\_1\Driver-State-Detection\original\_video2.mp4'  processed\_video\_path = r'C:\Users\mc2di\project\_1\Driver-State-Detection\processed\_video\_10FPS.mp4'  combined\_video\_path = r'C:\Users\mc2di\project\_1\Driver-State-Detection\combined\_video\_10FPS.mp4'  combine\_videos(original\_video\_path, processed\_video\_path, combined\_video\_path) |