**Gaze Eztimation**

import tensorflow as tf import cv2

import numpy as np import os

from sklearn.model\_selection import train\_test\_split import matplotlib.pyplot as plt

print("Imported Packages")

folder\_path = "MPIIFaceGaze\_preprocessed/Image/p00/face"

images = [img for img in os.listdir(folder\_path) if img.endswith(".jpg")] image\_data = []

for img\_name in images:

img\_path = os.path.join(folder\_path, img\_name) img = cv2.imread(img\_path)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) # Convert to RGB img = cv2.resize(img,(224,224))

image\_data.append(img)

print("Length of the Image : ",len(image\_data)) image\_data = np.array(image\_data)

print("Stored the data into an variable is successfully done")

train\_images, test\_images = train\_test\_split(image\_data, test\_size=0.2, random\_state=42) train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

print("Total Training Data: ",len(train\_images)) print("Total Testing Data : ",len(test\_images)) print("The images are sent for training and Testing")

DATA PROCESSING USING DUMMY DATA:

import torch

from torch.utils.data import Dataset, DataLoader import torch.nn as nn

import torch.nn.functional as F import numpy as np

# Custom Dataset

class FaceDataset(Dataset):

def init (self, face\_images, left\_eye\_images, right\_eye\_images): self.face\_images = face\_images

self.left\_eye\_images = left\_eye\_images self.right\_eye\_images = right\_eye\_images

self.gaze\_points = torch.randn(len(face\_images), 2) # Simulated targets self.gaze\_directions = torch.randn(len(face\_images), 2) # Simulated targets

def len (self):

return len(self.face\_images)

def getitem (self, idx): face\_img = self.face\_images[idx]

left\_eye\_img = self.left\_eye\_images[idx] right\_eye\_img = self.right\_eye\_images[idx]

# Convert NumPy to PyTorch tensor, if needed def to\_tensor(img):

return torch.tensor(img, dtype=torch.float32).permute(2, 0, 1) if isinstance(img, np.ndarray) else img

face\_img = to\_tensor(face\_img) left\_eye\_img = to\_tensor(left\_eye\_img) right\_eye\_img = to\_tensor(right\_eye\_img)

gaze\_point = self.gaze\_points[idx] gaze\_direction = self.gaze\_directions[idx]

return face\_img, left\_eye\_img, right\_eye\_img, gaze\_point, gaze\_direction

# Dummy Data

train\_face\_images = np.random.rand(100, 64, 64, 3)

train\_left\_eyes = np.random.rand(100, 64, 64, 3)

train\_right\_eyes = np.random.rand(100, 64, 64, 3)

test\_face\_images = np.random.rand(20, 64, 64, 3)

test\_left\_eyes = np.random.rand(20, 64, 64, 3)

test\_right\_eyes = np.random.rand(20, 64, 64, 3)

train\_dataset = FaceDataset(train\_face\_images, train\_left\_eyes, train\_right\_eyes) test\_dataset = FaceDataset(test\_face\_images, test\_left\_eyes, test\_right\_eyes)

train\_loader = DataLoader(train\_dataset, batch\_size=16, shuffle=True) test\_loader = DataLoader(test\_dataset, batch\_size=16)

sample\_batch = next(iter(train\_loader)) print(" Sample batch shapes:")

print("Face:", sample\_batch[0].shape) # [B, C, H, W]

print("Left Eye:", sample\_batch[1].shape) # [B, C, H, W] print("Right Eye:", sample\_batch[2].shape) # [B, C, H, W] print("Gaze point:", sample\_batch[3].shape) # [B, 2] print("Gaze direction:", sample\_batch[4].shape)

CONVOLUTIONAL NEURAL NETWORK MODAL TRAINING:

import torch

import torch.nn as nn

import torch.nn.functional as F

class GazeMultitaskNet(nn.Module): def init (self):

super(GazeMultitaskNet, self). init ()

self.output\_size = 64 # channels self.feature\_map\_size = 7 # (after adaptive pooling)

self.flatten\_dim = self.output\_size \* self.feature\_map\_size \* self.feature\_map\_size # 3136

self.conv\_face = self.\_conv\_block() self.conv\_left\_eye = self.\_conv\_block() self.conv\_right\_eye = self.\_conv\_block()

self.pool = nn.AdaptiveAvgPool2d((self.feature\_map\_size, self.feature\_map\_size))

self.attn = FocusAttentionLayer(self.flatten\_dim) self.fc = nn.Linear(self.flatten\_dim, 128) self.gaze\_point\_head = nn.Linear(128, 2) self.gaze\_direction\_head = nn.Linear(128, 2)

def \_conv\_block(self): return nn.Sequential(

nn.Conv2d(3, 32, kernel\_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(2, 2),

nn.Conv2d(32, 64, kernel\_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(2, 2),

)

def forward(self, face\_img, left\_eye\_img, right\_eye\_img):

face\_feat = self.pool(self.conv\_face(face\_img)) # [B, 64, 7, 7]

left\_feat = self.pool(self.conv\_left\_eye(left\_eye\_img)) # [B, 64, 7, 7]

right\_feat = self.pool(self.conv\_right\_eye(right\_eye\_img)) # [B, 64, 7, 7]

# Flatten

face\_feat = face\_feat.view(face\_feat.size(0), -1) # [B, 3136] left\_feat = left\_feat.view(left\_feat.size(0), -1)

right\_feat = right\_feat.view(right\_feat.size(0), -1)

eye\_feat = (left\_feat + right\_feat) / 2

attended\_feat = self.attn(face\_feat, eye\_feat) # [B, 3136]

x = F.relu(self.fc(attended\_feat)) gaze\_point = self.gaze\_point\_head(x)

gaze\_direction = self.gaze\_direction\_head(x) print(" Gaze Point : ",gaze\_point) print("Gaze Direction : ",gaze\_direction)

return gaze\_point, gaze\_direction

FOCUS ATTENTION LAYER:

class FocusAttentionLayer(nn.Module): def init (self, feature\_dim):

super(FocusAttentionLayer, self). init () self.fc\_face = nn.Linear(feature\_dim, feature\_dim) self.fc\_eye = nn.Linear(feature\_dim, feature\_dim)

def forward(self, face\_feat, eye\_feat):

# Project features

face\_proj = self.fc\_face(face\_feat) # [B, D] eye\_proj = self.fc\_eye(eye\_feat)# [B, D]

# Dot product attention score

attention\_score = torch.sum(face\_proj \* eye\_proj, dim=1, keepdim=True) # [B, 1]

attention\_weight = torch.sigmoid(attention\_score) # use sigmoid instead of softmax for scalar weights

# Weighted fusion

attended\_feat = attention\_weight \* eye\_feat + (1 - attention\_weight) \* face\_feat

return attended\_feat

INITIALISE LOSS FUNCTION, MODAL, AND OPTIMIZER:

# Initialize model, loss function, and optimizer

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu") modal = GazeMultitaskNet().to(device)

print("GazeMultitaskNet Initialized on", device)

criterion = nn.MSELoss()

optimizer = torch.optim.Adam(modal.parameters(), lr=0.001)

# Training loop # Training loop

num\_epochs = 10

for epoch in range(num\_epochs): modal.train()

running\_loss = 0.0

for face\_img, left\_eye\_img, right\_eye\_img, target\_point, target\_direction in train\_loader:

# Move data to device (GPU/CPU) face\_img = face\_img.to(device) left\_eye\_img = left\_eye\_img.to(device) right\_eye\_img = right\_eye\_img.to(device) target\_point = target\_point.to(device)

target\_direction = target\_direction.to(device)

# ⬛“ Corrected model input

pred\_point, pred\_direction = modal(face\_img, left\_eye\_img, right\_eye\_img)

# Compute losses for both tasks

loss\_point = criterion(pred\_point, target\_point) loss\_direction = criterion(pred\_direction, target\_direction)

# Total multitask loss

loss = loss\_point + loss\_direction

# Backpropagation optimizer.zero\_grad() loss.backward() optimizer.step()

running\_loss += loss.item()

avg\_loss = running\_loss / len(train\_loader)

print(f"ç ? Epoch [{epoch + 1}/{num\_epochs}], ⬛“ Loss: {avg\_loss:.4f}") print("Training Completed")

OPENCV :

import cv2 import torch

import numpy as np import os

import time # for tracking every second

# Ensure your model is imported correctly

# Device

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Load model

modal = GazeMultitaskNet().to(device) model\_path = "gaze\_model.pth"

if os.path.exists(model\_path): modal.load\_state\_dict(torch.load(model\_path, map\_location=device)) print("Pretrained model loaded successfully.")

else:

print(" ị Model file not found. Using untrained model.") modal.eval()

# Haar cascades

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

eye\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_eye.xml')

# Video capture

cap = cv2.VideoCapture(0)

def preprocess\_roi(roi):

roi = cv2.resize(roi, (64, 64))

roi = roi.astype(np.float32) / 255.0

roi = torch.tensor(roi).permute(2, 0, 1).unsqueeze(0) # [1, 3, 64, 64] return roi.to(device)

# Timer for logging every second last\_logged\_time = time.time()

while True:

ret, frame = cap.read() if not ret:

break

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) faces = face\_cascade.detectMultiScale(gray, 1.3, 5)

for (x, y, w, h) in faces:

face\_roi = frame[y:y+h, x:x+w]

eyes = eye\_cascade.detectMultiScale(gray[y:y+h, x:x+w])

if len(eyes) >= 2:

eyes = sorted(eyes, key=lambda e: e[0]) ex1, ey1, ew1, eh1 = eyes[0]

ex2, ey2, ew2, eh2 = eyes[1]

left\_eye = frame[y+ey1:y+ey1+eh1, x+ex1:x+ex1+ew1] right\_eye = frame[y+ey2:y+ey2+eh2, x+ex2:x+ex2+ew2]

face\_tensor = preprocess\_roi(face\_roi) left\_tensor = preprocess\_roi(left\_eye) right\_tensor = preprocess\_roi(right\_eye)

with torch.no\_grad():

gaze\_point, gaze\_dir = modal(face\_tensor, left\_tensor, right\_tensor) gaze\_point = gaze\_point.squeeze().cpu().numpy()

gaze\_dir = gaze\_dir.squeeze().cpu().numpy()

center\_x, center\_y = int(x + w/2), int(y + h/2) point\_x = int(center\_x + gaze\_point[0]\*50) point\_y = int(center\_y + gaze\_point[1]\*50) dir\_x = int(center\_x + gaze\_dir[0]\*50)

dir\_y = int(center\_y + gaze\_dir[1]\*50)

cv2.arrowedLine(frame, (center\_x, center\_y), (dir\_x, dir\_y), (0, 255, 0),

cv2.circle(frame, (point\_x, point\_y), 5, (0, 0, 255), -1)

cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

# Get absolute center coordinates of eyes

left\_eye\_center = (x + ex1 + ew1 // 2, y + ey1 + eh1 // 2) right\_eye\_center = (x + ex2 + ew2 // 2, y + ey2 + eh2 // 2)

# Print every 1 second current\_time = time.time()

if current\_time - last\_logged\_time >= 1:

print(f"👁 Left Eye: X={left\_eye\_center[0]} Y={left\_eye\_center[1]}")

print(f"👁 Right Eye: X={right\_eye\_center[0]} Y={right\_eye\_center[1]}")

last\_logged\_time = current\_time

# Create a second blank canvas (white screen)

white\_screen = np.ones\_like(frame) \* 255 # A blank white canvas

# Draw X and Y points on the second screen

cv2.circle(white\_screen, (left\_eye\_center[0], left\_eye\_center[1]), 5, (0, 0, 255), -1) # Red point for X coordinate

cv2.circle(white\_screen, (right\_eye\_center[0], right\_eye\_center[1]), 5, (255, 0, 0), -1) # Blue point for Y coordinate

# Show both the gaze tracking frame and the white screen

combined = np.hstack((frame, white\_screen)) # Combine the two images side by side

cv2.imshow('Gaze Estimation UI (Left) & Coordinates (Right)', combined)

if cv2.waitKey(1) == 27: # Press Esc to exit

break

cap.release() cv2.destroyAllWindows()

print(" Gaze Estimation Completed Successfully")

# Save the model torch.save(modal.state\_dict(), model\_path) print(" Model Saved Successfully")