ML-Features-drowsiness

November 26, 2025

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[2]:
```
cd /content/drive/MyDrive/Drowsiness-Detection
```

```
/content/drive/MyDrive/Drowsiness-Detection
```

[3]:
```
!ls
```

```
drowsiness-main-pipeline.ipynb  ML-Features-drowsiness.ipynb
```

[ ]:
```

```

[ ]:
```
import random
import numpy as np
import tensorflow as tf

SEED = 42

random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)

# !rm -rf /kaggle/working/*  ---> this for clearing the output
```

[ ]:
```
# No need to run this block

# !pip install protobuf==3.20.3 --force-reinstall --quiet
# !pip install mediapipe --force-reinstall --quiet

# import google.protobuf
# print(google.protobuf.__version__)

# # 3.20.3
# !pip install facenet-pytorch --quiet
```

[ ]:
```

```

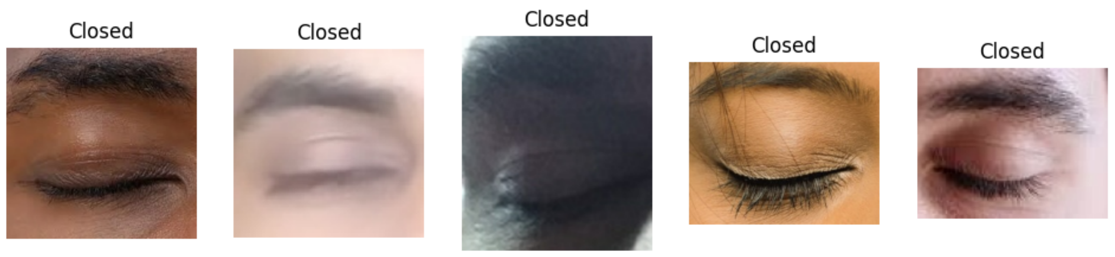
[ ]:
```
import tensorflow as tf
print(tf.__version__)
```

```
2.18.0
```

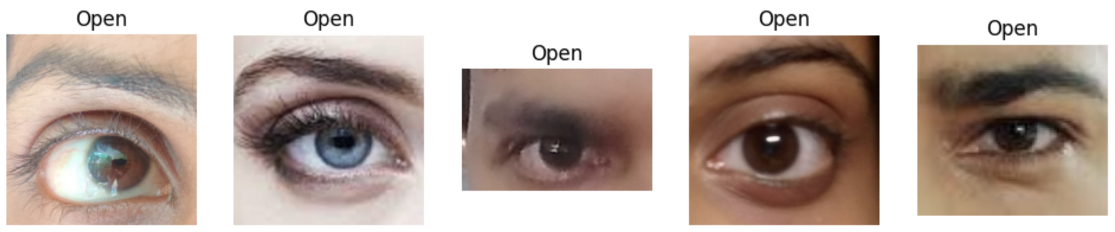
[ ]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2


# dataset path
data_dir = '/kaggle/input/yawdd-cwe-glasses/yawdd,cwe,glasses-dataset/train'

# folders inside train
folders = ['Closed', 'Open', 'yawn', 'no_yawn']

# dictionary to store counts
image_counts = {}

# loop through folders
for folder in folders:
    folder_path = os.path.join(data_dir, folder)
    images = os.listdir(folder_path)
    image_counts[folder] = len(images)

    print(f"\nFolder: {folder}")
    print(f"Number of images: {len(images)}")

    # plot 5 sample images
    plt.figure(figsize=(12, 6))
    for i, img_name in enumerate(images[:5]):
        img_path = os.path.join(folder_path, img_name)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.subplot(1, 5, i+1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(folder)
    plt.show()
```

```python
    # check image sizes
    sizes = set()
    for img_name in images:
        img_path = os.path.join(folder_path, img_name)
        img = cv2.imread(img_path)
        if img is not None:
            sizes.add(img.shape[:2])  # (height, width)

    if len(sizes) == 1:
        print(f" All images in {folder} are of the same size: {list(sizes)[0]}")
    else:
        print(f" Images in {folder} are of different sizes")
```

```
Folder: Closed
Number of images: 660
```



```
  Images in Closed are of different sizes

Folder: Open
Number of images: 663
```



```
  Images in Open are of different sizes

Folder: yawn
Number of images: 723
```

```
 All images in yawn are of the same size: (480, 640)

Folder: no_yawn
Number of images: 725
```

```
 All images in no_yawn are of the same size: (480, 640)
```

0.0.1 Face Crop Using HAAR Cascade model Method 1

```
[ ]: import cv2
     import os
     from pathlib import Path

     # Paths to Kaggle Working Directory (Writable)
     working_dir = '/kaggle/working'
     yawn_crop_dir = os.path.join(working_dir, 'yawnCrop')
     no_yawn_crop_dir = os.path.join(working_dir, 'no_yawnCrop')

     # Create the new directories in the working directory
     Path(yawn_crop_dir).mkdir(parents=True, exist_ok=True)
     Path(no_yawn_crop_dir).mkdir(parents=True, exist_ok=True)

     # Load OpenCV's Haar Cascade for face detection
     face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +␣
      ↪'haarcascade_frontalface_default.xml')

     def crop_and_save_faces(source_dir, destination_dir):
         # Loop through all images in the source directory
         for img_name in os.listdir(source_dir):
             img_path = os.path.join(source_dir, img_name)
```

```
        # Read the image (in color format)
        img = cv2.imread(img_path)

        # Detect faces in color image
        faces = face_cascade.detectMultiScale(img, scaleFactor=1.1,␣
↪minNeighbors=5, minSize=(30, 30))

        # If faces are detected, crop and save them
        for i, (x, y, w, h) in enumerate(faces):
            # Crop the face from the color image
            face = img[y:y+h, x:x+w]

            # Save the cropped face in the destination folder
            crop_name = f"{os.path.splitext(img_name)[0]}_face_{i+1}.jpg"
            crop_path = os.path.join(destination_dir, crop_name)
            cv2.imwrite(crop_path, face)

# Example paths to your original images (these should still be in /kaggle/input/
↪)
yawn_dir = '/kaggle/input/yawdd-cwe-glasses/yawdd,cwe,glasses-dataset/train/
↪yawn'
no_yawn_dir = '/kaggle/input/yawdd-cwe-glasses/yawdd,cwe,glasses-dataset/train/
↪no_yawn'

# Crop and save faces for 'yawn' images
crop_and_save_faces(yawn_dir, yawn_crop_dir)

# Crop and save faces for 'no_yawn' images
crop_and_save_faces(no_yawn_dir, no_yawn_crop_dir)

print("Face cropping and saving completed!")
```

```
Face cropping and saving completed!
```

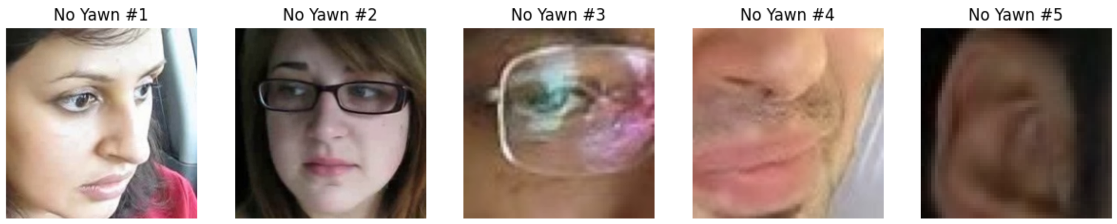
[ ]:
```
import os
import random
import cv2
import matplotlib.pyplot as plt



# Function to show 5 random images from a folder
def show_random_images(folder_path, title, num_images=5):
    image_files = os.listdir(folder_path)
    selected_images = random.sample(image_files, num_images)

    plt.figure(figsize=(15, 5))
    for i, img_name in enumerate(selected_images):
        img_path = os.path.join(folder_path, img_name)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB for␣
↪matplotlib

        plt.subplot(1, num_images, i+1)
        plt.imshow(img)
        plt.title(f"{title} #{i+1}")
        plt.axis('off')
    plt.suptitle(f"Random {num_images} Images from {title} Class", fontsize=16)
    plt.show()



def check_same_shape(folder_path):
    image_shapes = []
    image_files = [f for f in os.listdir(folder_path) if f.lower().endswith(('.
↪png', '.jpg', '.jpeg'))]

    for img_file in image_files:
        img_path = os.path.join(folder_path, img_file)
        img = cv2.imread(img_path)
        if img is None:
            print(f"Warning: Unable to load {img_file}")
            continue
        image_shapes.append(img.shape)

    if len(image_shapes) == 0:
        print(f"No valid images found in {folder_path}")
        return

    first_shape = image_shapes[0]
    all_same = all(shape == first_shape for shape in image_shapes)

    print(f"All images in '{os.path.basename(folder_path)}' have the same shape?
↪ {'Yes' if all_same else 'No'}")
```

[ ]:
```
# Paths to cropped image folders in /kaggle/working
yawn_crop_dir = '/kaggle/working/yawnCrop'
no_yawn_crop_dir = '/kaggle/working/no_yawnCrop'


# Show 5 random images from each class
show_random_images(yawn_crop_dir, 'Yawn')
show_random_images(no_yawn_crop_dir, 'No Yawn')
```

```
# Folder paths
yawn_crop_dir = '/kaggle/working/yawnCrop'
no_yawn_crop_dir = '/kaggle/working/no_yawnCrop'

# Count images and check shape consistency
print(f"Number of images in '{os.path.basename(yawn_crop_dir)}': {len(os.
  ↪listdir(yawn_crop_dir))}")
check_same_shape(yawn_crop_dir)

print(f"Number of images in '{os.path.basename(no_yawn_crop_dir)}': {len(os.
  ↪listdir(no_yawn_crop_dir))}")
check_same_shape(no_yawn_crop_dir)


# This model not work very good, and also skips many images because of not able␣
  ↪to find face in them
```

Random 5 Images from Yawn Class

Random 5 Images from No Yawn Class



```
Number of images in 'yawnCrop': 497
All images in 'yawnCrop' have the same shape? No
Number of images in 'no_yawnCrop': 469
All images in 'no_yawnCrop' have the same shape? No
```

[ ]:

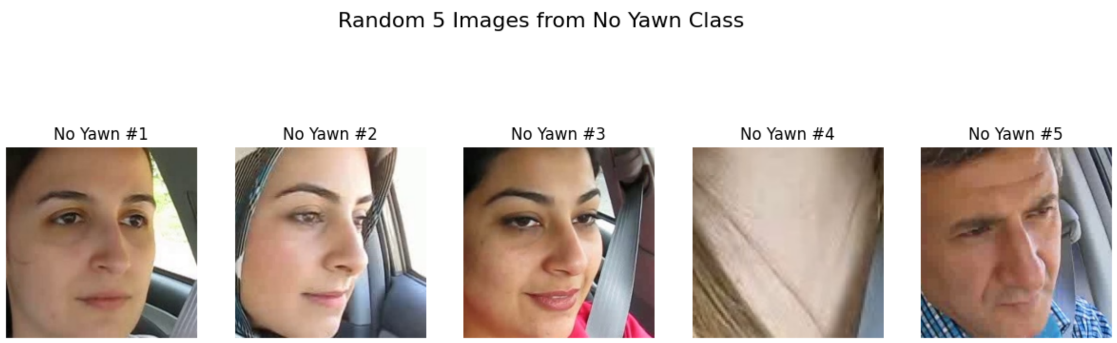
[ ]:

0.0.2 FACE CROP Method 2 HAAR

[ ]:
```python
import os
import cv2
from tqdm import tqdm

# === 1. Setup paths ===

# Input (read-only)
data_dir = '/kaggle/input/yawdd-cwe-glasses/yawdd,cwe,glasses-dataset/train'
yawn_dir = os.path.join(data_dir, 'yawn')
no_yawn_dir = os.path.join(data_dir, 'no_yawn')

# Output (writable, inside working directory)
output_base = '/kaggle/working/train_cropped'
yawn_crop_dir = os.path.join(output_base, 'yawnCrop')
no_yawn_crop_dir = os.path.join(output_base, 'no_yawnCrop')

os.makedirs(yawn_crop_dir, exist_ok=True)
os.makedirs(no_yawn_crop_dir, exist_ok=True)

# === 2. Load OpenCV face detector ===
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +␣
↪'haarcascade_frontalface_default.xml')

# === 3. Define function to crop face ===

def crop_face(image_path, save_path):
    img = cv2.imread(image_path)
    if img is None:
        print(f"Failed to load {image_path}")
        return False

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)

    if len(faces) == 0:
        print(f"No face found in {image_path}, skipping.")
        return False

    # Crop first detected face
    (x, y, w, h) = faces[0]
    face_img = img[y:y+h, x:x+w]
    # Save cropped face image
    cv2.imwrite(save_path, face_img)
    return True

# === 4. Process folders and crop images ===

def process_folder(input_folder, output_folder):
    filenames = [f for f in os.listdir(input_folder) if f.lower().endswith(('.
 ↪png', '.jpg', '.jpeg'))]
    for filename in tqdm(filenames, desc=f"Processing {os.path.
 ↪basename(input_folder)}"):
        input_path = os.path.join(input_folder, filename)
        output_path = os.path.join(output_folder, filename)
        crop_face(input_path, output_path)

process_folder(yawn_dir, yawn_crop_dir)
process_folder(no_yawn_dir, no_yawn_crop_dir)


# This basic HAAR model is not so good in face cropping, and it skips many␣
 ↪images because it cant able to find faces in those ...
```

[ ]:
```
import os
import random
import cv2
import matplotlib.pyplot as plt

# Paths to cropped image folders in /kaggle/working
yawn_crop_dir = '/kaggle/working/train_cropped/yawnCrop'
no_yawn_crop_dir = '/kaggle/working/train_cropped/no_yawnCrop'



# Show 5 random images from each class
show_random_images(yawn_crop_dir, 'Yawn')
show_random_images(no_yawn_crop_dir, 'No Yawn')



# Folder paths
yawn_crop_dir = '/kaggle/working/train_cropped/yawnCrop'
no_yawn_crop_dir = '/kaggle/working/train_cropped/no_yawnCrop'

# Count images and check shape consistency
print(f"Number of images in '{os.path.basename(yawn_crop_dir)}': {len(os.
 ↪listdir(yawn_crop_dir))}")
```

```
check_same_shape(yawn_crop_dir)

print(f"Number of images in '{os.path.basename(no_yawn_crop_dir)}': {len(os.
↪listdir(no_yawn_crop_dir))}")
check_same_shape(no_yawn_crop_dir)
```



```
Number of images in 'yawnCrop': 421
All images in 'yawnCrop' have the same shape? No
Number of images in 'no_yawnCrop': 416
All images in 'no_yawnCrop' have the same shape? No
```

[ ]:

```

```

0.0.3 Method 3 MTCNN facenet pytorch BEST For Face Crop

[ ]:

```
import os
from tqdm import tqdm
from PIL import Image
import cv2
from facenet_pytorch import MTCNN
import torch
```

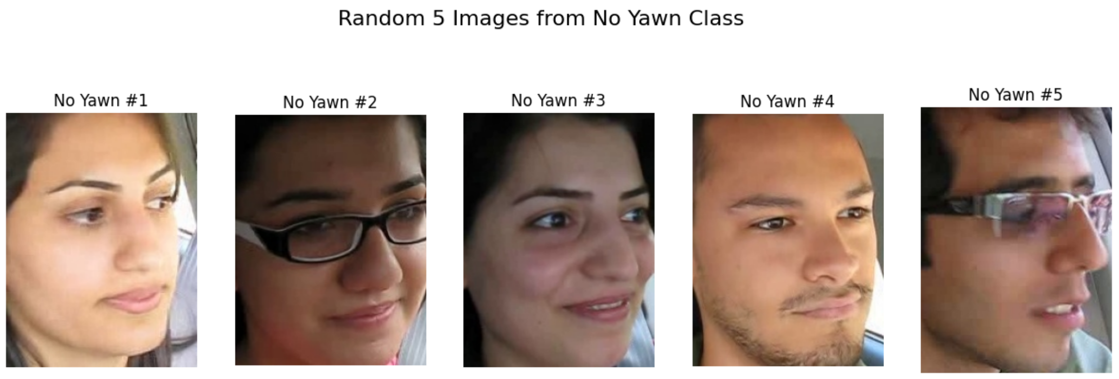
```python
def load_image_cv2_to_pil(image_path):
    img_bgr = cv2.imread(image_path)
    if img_bgr is None:
        raise ValueError(f"Failed to load image {image_path} with cv2")
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
    pil_img = Image.fromarray(img_rgb)
    return pil_img

device = 'cuda' if torch.cuda.is_available() else 'cpu'
mtcnn = MTCNN(keep_all=False, device=device)

import numpy as np
import cv2

def crop_face_mtcnn_cv2(image_path, save_path):
    try:
        img = load_image_cv2_to_pil(image_path)
    except Exception as e:
        print(f"Skipping corrupted/unreadable image: {image_path}. Error: {e}")
        return False

    boxes, _ = mtcnn.detect(img)
    if boxes is None:
        print(f"No face detected in {image_path}")
        return False

    box = boxes[0]
    width, height = img.size

    left = max(int(box[0]), 0)
    top = max(int(box[1]), 0)
    right = min(int(box[2]), width)
    bottom = min(int(box[3]), height)

    if right <= left or bottom <= top:
        print(f"Invalid bounding box for {image_path}, skipping.")
        return False

    face_img = img.crop((left, top, right, bottom))

    # Convert PIL Image to OpenCV format for saving
    face_np = np.array(face_img)          # RGB format
    face_bgr = cv2.cvtColor(face_np, cv2.COLOR_RGB2BGR)

    # Save with OpenCV
    success = cv2.imwrite(save_path, face_bgr)
    if not success:
        print(f"Failed to save cropped image: {save_path}")
        return False

    return True



def process_folder_mtcnn_cv2(input_folder, output_folder):
    os.makedirs(output_folder, exist_ok=True)
    filenames = [f for f in os.listdir(input_folder) if f.lower().endswith(('.
↪png', '.jpg', '.jpeg'))]

    for filename in tqdm(filenames, desc=f"Processing {os.path.
↪basename(input_folder)}"):
        input_path = os.path.join(input_folder, filename)
        output_path = os.path.join(output_folder, filename)
        crop_face_mtcnn_cv2(input_path, output_path)

data_dir = '/kaggle/input/yawdd-cwe-glasses/yawdd,cwe,glasses-dataset/train'
yawn_dir = os.path.join(data_dir, 'yawn')
no_yawn_dir = os.path.join(data_dir, 'no_yawn')

output_base = '/kaggle/working/train_cropped_mtcnn'
yawn_crop_dir = os.path.join(output_base, 'yawnCrop')
no_yawn_crop_dir = os.path.join(output_base, 'no_yawnCrop')

process_folder_mtcnn_cv2(yawn_dir, yawn_crop_dir)
process_folder_mtcnn_cv2(no_yawn_dir, no_yawn_crop_dir)
```

```
Processing yawn: 100%|      | 723/723 [00:28<00:00, 24.96it/s]
Processing no_yawn: 100%|      | 725/725 [00:27<00:00, 26.19it/s]
```

[ ]:

[ ]:
```
import os
import random
import cv2
import matplotlib.pyplot as plt

# Paths to cropped image folders in /kaggle/working
yawn_crop_dir = '/kaggle/working/train_cropped_mtcnn/yawnCrop'
no_yawn_crop_dir = '/kaggle/working/train_cropped_mtcnn/no_yawnCrop'


# Show 5 random images from each class
show_random_images(yawn_crop_dir, 'Yawn')
show_random_images(no_yawn_crop_dir, 'No Yawn')
```

```python
# Folder paths
yawn_crop_dir = '/kaggle/working/train_cropped_mtcnn/yawnCrop'
no_yawn_crop_dir = '/kaggle/working/train_cropped_mtcnn/no_yawnCrop'

# Count images and check shape consistency
print(f"Number of images in '{os.path.basename(yawn_crop_dir)}': {len(os.
↪listdir(yawn_crop_dir))}")
check_same_shape(yawn_crop_dir)

print(f"Number of images in '{os.path.basename(no_yawn_crop_dir)}': {len(os.
↪listdir(no_yawn_crop_dir))}")
check_same_shape(no_yawn_crop_dir)


# This Method works perfectly
```



```
Number of images in 'yawnCrop': 723
```

```
All images in 'yawnCrop' have the same shape? No
Number of images in 'no_yawnCrop': 725
All images in 'no_yawnCrop' have the same shape? No
```

[ ]:

```

```

[ ]:

```

```

[ ]:

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import confusion_matrix, classification_report
import itertools
import seaborn as sns
import pathlib

# -------------------------
# User config / hyperparams
# -------------------------
data_dir = '/kaggle/working/train_cropped_mtcnn'  # CHANGE if needed
TARGET_CLASSES = ['yawnCrop', 'no_yawnCrop']
IMG_SIZE = (224, 224)        # fixed size for both custom and transfer models
BATCH_SIZE = 32
SEED = 42
EPOCHS_CUSTOM = 50
EPOCHS_TL = 20
LEARNING_RATE = 1e-3
```

[ ]:

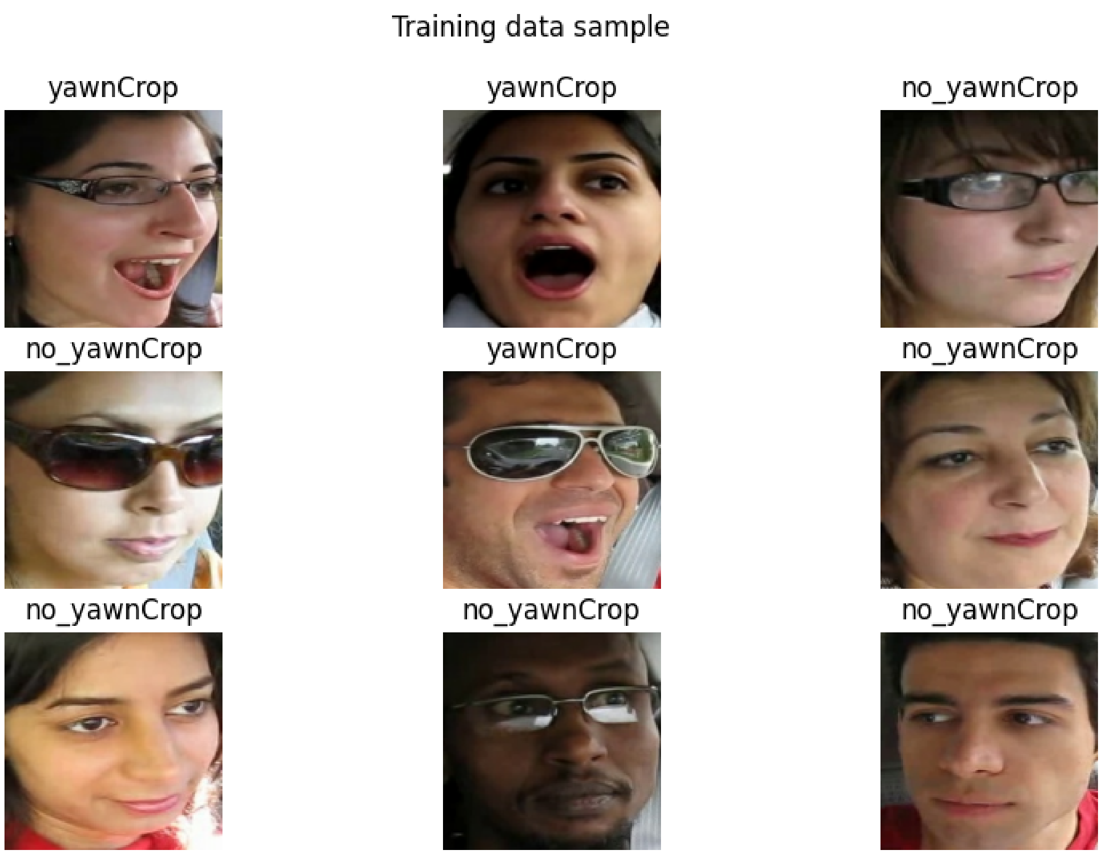
```
# -------------------------
# Verify dataset & counts
# -------------------------
for c in TARGET_CLASSES:
    p = os.path.join(data_dir, c)
    if not os.path.isdir(p):
        raise FileNotFoundError(f"Expected directory for class '{c}' at: {p}")
    count = len([f for f in os.listdir(p) if os.path.isfile(os.path.join(p,␣
↪f))])
    print(f"Class '{c}': {count} images")
```

```
Class 'yawnCrop': 723 images
Class 'no_yawnCrop': 725 images
```

[ ]:

```
# 1. Load datasets
train_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    labels="inferred",
    label_mode="binary",
    class_names=["yawnCrop", "no_yawnCrop"],
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    validation_split=0.3,   # 70% train, 30% val+test
    subset="training",
    seed=SEED
)

val_test_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    labels="inferred",
    label_mode="binary",
    class_names=["yawnCrop", "no_yawnCrop"],
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    validation_split=0.3,
    subset="validation",
    seed=SEED
)

# 2. Save class names BEFORE caching/prefetch
class_names = train_ds.class_names
print("Class names (inferred):", class_names)

# 3. Split val_test_ds into val/test
val_batches = int(0.5 * tf.data.experimental.cardinality(val_test_ds).numpy())
val_ds = val_test_ds.take(val_batches)
test_ds = val_test_ds.skip(val_batches)

print("\nDataset sizes (batches):")
print("Train:", tf.data.experimental.cardinality(train_ds).numpy())
print("Val:", tf.data.experimental.cardinality(val_ds).numpy())
print("Test:", tf.data.experimental.cardinality(test_ds).numpy())

# 4. Now apply cache/prefetch (after saving class_names)
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)

# -------------------------
# Optional: visualize some samples
# -------------------------
def show_batch(dataset, title="Sample images"):
    plt.figure(figsize=(10, 6))
    for images, labels in dataset.take(1):
        for i in range(min(9, images.shape[0])):
            ax = plt.subplot(3, 3, i + 1)
            plt.imshow(images[i].numpy().astype("uint8"))
            label = int(labels[i].numpy())  # float→int
            plt.title(class_names[label])
            plt.axis("off")
    plt.suptitle(title)
    plt.show()

show_batch(train_ds, "Training data sample")
```

```
Found 1448 files belonging to 2 classes.
Using 1014 files for training.

I0000 00:00:1759262881.953792      36 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 13740 MB memory:  -> device:
0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5
I0000 00:00:1759262881.954604      36 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:1 with 13942 MB memory:  -> device:
1, name: Tesla T4, pci bus id: 0000:00:05.0, compute capability: 7.5

Found 1448 files belonging to 2 classes.
Using 434 files for validation.
Class names (inferred): ['yawnCrop', 'no_yawnCrop']

Dataset sizes (batches):
Train: 32
Val: 7
Test: 7

/tmp/ipykernel_36/954708821.py:55: DeprecationWarning: Conversion of an array
with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you
extract a single element from your array before performing this operation.
(Deprecated NumPy 1.25.)
  label = int(labels[i].numpy())  # float→int
```



0.0.4 Check for Class Imbalance

[ ]:
```python
# 1. Check distribution of labels
def count_labels(dataset):
    counts = {0: 0, 1: 0}
    for _, labels in dataset.unbatch():
        lbl = int(labels.numpy())
        counts[lbl] += 1
    return counts

print("Train label distribution:", count_labels(train_ds))
print("Val label distribution:", count_labels(val_ds))
print("Test label distribution:", count_labels(test_ds))
```

```
/tmp/ipykernel_36/1358632873.py:5: DeprecationWarning: Conversion of an array
with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you
extract a single element from your array before performing this operation.
(Deprecated NumPy 1.25.)
  lbl = int(labels.numpy())
```

```
Train label distribution: {0: 518, 1: 496}
```

```
Val label distribution: {0: 111, 1: 113}
Test label distribution: {0: 94, 1: 116}
```

[ ]:
```
# 2. Compute Class weight

from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# Collect all training labels
train_labels = []
for _, labels in train_ds.unbatch():
    train_labels.append(int(labels.numpy()))
train_labels = np.array(train_labels)

# Compute weights
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(train_labels),
    y=train_labels
)
class_weights = dict(enumerate(class_weights))
print("Class Weights:", class_weights)
```

```
/tmp/ipykernel_36/1445992484.py:9: DeprecationWarning: Conversion of an array
with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you
extract a single element from your array before performing this operation.
(Deprecated NumPy 1.25.)
  train_labels.append(int(labels.numpy()))
```

```
Class Weights: {0: 0.9787644787644788, 1: 1.0221774193548387}
```

[ ]:
```

```

[ ]:
```

```

[ ]:
```
# -------------------------
# Data augmentation + preprocessing layers
# -------------------------
data_augmentation = keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.08),
    layers.RandomZoom(0.08),
    layers.RandomContrast(0.06),
], name="data_augmentation")

# Normalization layer (rescale pixels 0-1)
rescale = layers.Rescaling(1./255)
```

[ ]:
```python
# print(train_ds.shape)
print(type(train_ds))
print(train_ds)

# print(val_ds.shape)
print(type(val_ds))
print(val_ds)
```

```
<class 'tensorflow.python.data.ops.prefetch_op._PrefetchDataset'>
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None, 1), dtype=tf.float32,
name=None))>
<class 'tensorflow.python.data.ops.prefetch_op._PrefetchDataset'>
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None, 1), dtype=tf.float32,
name=None))>
```

[ ]:
```python
import sys
print(sys.version)
```

```
3.11.13 (main, Jun  4 2025, 08:57:29) [GCC 11.4.0]
```

[ ]:
```python
# Build y_true and y_pred arrays for confusion matrix & classification report
def get_labels_and_predictions(model, dataset):
    y_true = []
    y_pred = []
    for images, labels in dataset.unbatch().batch(256):  # manageable chunks
        preds = model.predict(images, verbose=0).ravel()
        y_pred.extend((preds >= 0.5).astype(int).tolist())
        y_true.extend(labels.numpy().astype(int).tolist())
    return np.array(y_true), np.array(y_pred)


# Plot training curves for custom model
def plot_history(history, title_prefix=""):
    history_dict = history.history
    epochs_range = range(1, len(history_dict['loss']) + 1)

    plt.figure(figsize=(12, 4))
    # Loss
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, history_dict['loss'], label='train_loss')
    plt.plot(epochs_range, history_dict['val_loss'], label='val_loss')
    plt.title(f'{title_prefix} Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    # Accuracy
    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, history_dict['accuracy'], label='train_acc')
    plt.plot(epochs_range, history_dict['val_accuracy'], label='val_acc')
    plt.title(f'{title_prefix} Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.show()
```

[ ]:

```
[ ]: import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import cv2
from tensorflow import keras

#  Grad-CAM heatmap function for custom CNN
def make_gradcam_heatmap(img_array, model, last_conv_layer_name,␣
 ↪pred_index=None):
    grad_model = tf.keras.models.Model(
        [model.inputs],
        [model.get_layer(last_conv_layer_name).output, model.output]
    )

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(predictions[0])
        class_channel = predictions[:, pred_index]

    # Compute gradients
    grads = tape.gradient(class_channel, conv_outputs)

    # Compute pooled grads
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # Weight conv outputs with grads
    conv_outputs = conv_outputs[0]
    heatmap = conv_outputs @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)

    # Normalize
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
```

```python
    return heatmap.numpy()

#  Overlay heatmap on image
def overlay_gradcam(img, heatmap, alpha=0.4, colormap=cv2.COLORMAP_JET):
    # Ensure image is uint8
    if img.dtype != np.uint8:
        img = np.uint8(255 * img / np.max(img))

    # Resize heatmap to match input image
    heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))

    # Convert heatmap to RGB
    heatmap = np.uint8(255 * heatmap)
    heatmap = cv2.applyColorMap(heatmap, colormap)
    heatmap = cv2.cvtColor(heatmap, cv2.COLOR_BGR2RGB)  # convert from BGR → RGB

    # Overlay heatmap on image
    superimposed_img = cv2.addWeighted(heatmap, alpha, img, 1 - alpha, 0)
    return superimposed_img


def show_gradcam_samples(dataset, class_names, model, last_conv_layer_name,␣
 ↪num_images=3):
    plt.figure(figsize=(12, 6))
    count = 0
    for images, labels in dataset.take(1):  # take 1 batch
        for i in range(min(num_images, images.shape[0])):
            img = images[i].numpy().astype("uint8")
            label = int(labels[i].numpy())  #  cast to int

            img_array = np.expand_dims(images[i], axis=0)
            heatmap = make_gradcam_heatmap(img_array, model,␣
 ↪last_conv_layer_name)
            superimposed_img = overlay_gradcam(img, heatmap)

            # Show original
            plt.subplot(2, num_images, i + 1)
            plt.imshow(img)
            plt.title(f"True: {class_names[label]}")
            plt.axis("off")

            # Show GradCAM
            plt.subplot(2, num_images, i + 1 + num_images)
            plt.imshow(superimposed_img)
            plt.title("GradCAM")
            plt.axis("off")
            count += 1
        break  # only take one batch
    plt.tight_layout()
    plt.show()
```

[ ]:

```

```

0.1 CNN V2 with modification

[ ]:

```
def cbam_block(input_feature, ratio=8):
    """Convolutional Block Attention Module (CBAM)"""

    channel = input_feature.shape[-1]

    # -------- Channel Attention --------
    avg_pool = layers.GlobalAveragePooling2D()(input_feature)
    max_pool = layers.GlobalMaxPooling2D()(input_feature)

    # Shared MLP
    shared_dense_1 = layers.Dense(channel // ratio, activation='relu') # ( 2␣
 ↪layers first contains c/r unit)
    shared_dense_2 = layers.Dense(channel)                            #␣
 ↪(second layer contains c unit)

    avg_out = shared_dense_2(shared_dense_1(avg_pool))
    max_out = shared_dense_2(shared_dense_1(max_pool))

    channel_attention = layers.Add()([avg_out, max_out])
    channel_attention = layers.Activation('sigmoid')(channel_attention)
    channel_attention = layers.Reshape((1, 1, channel))(channel_attention)

    x = layers.Multiply()([input_feature, channel_attention])

    # -------- Spatial Attention --------
    avg_pool_spatial = layers.Lambda(lambda t: K.mean(t, axis=-1,␣
 ↪keepdims=True))(x)
    max_pool_spatial = layers.Lambda(lambda t: K.max(t, axis=-1,␣
 ↪keepdims=True))(x)
    concat = layers.Concatenate(axis=-1)([avg_pool_spatial, max_pool_spatial])
    spatial_attention = layers.Conv2D(1, kernel_size=7, padding='same',␣
 ↪activation='sigmoid')(concat)
    x = layers.Multiply()([x, spatial_attention])

    return x

def build_custom_cnn_cbam_v2(input_shape=(224,224,3)):
    inputs = keras.Input(shape=input_shape)

    #  Data Augmentation + Normalization
    x = data_augmentation(inputs)
    x = rescale(x)

    # ----- Conv Block 1 -----
    x = layers.Conv2D(32, 3, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D()(x)

    # ----- Conv Block 2 -----
    x = layers.Conv2D(64, 3, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D()(x)

    # ----- Conv Block 3 -----
    x = layers.Conv2D(128, 3, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D()(x)

    # ----- Conv Block 4 With CBAM -----
    x = layers.Conv2D(256, 3, padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = cbam_block(x)
    x = layers.MaxPooling2D()(x)

    #   Global Pooling (both Avg + Max)
    gap = layers.GlobalAveragePooling2D()(x)
    gmp = layers.GlobalMaxPooling2D()(x)
    x = layers.Concatenate()([gap, gmp])

    #   Dense Layers
    x = layers.Dense(256, activation='relu')(x)
    x = layers.Dropout(0.25)(x)

    outputs = layers.Dense(1, activation='sigmoid')(x)

    model = keras.Model(inputs, outputs, name='custom_cnn_cbam_v2')
    return model
```

[ ]:
```
improved_model = build_improved_cnn()
improved_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-2),  # slightly lower lr
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

```
improved_model.summary()
```

[ ]:

```
```

[ ]:

```
```

0.1.1 USE EAR & MAR and USE ML MODEL on image

[ ]:

```
import os
import cv2
import dlib
import numpy as np
from scipy.spatial import distance as dist
from imutils import face_utils
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# --- 1. Paths ---
base_dir = '/kaggle/working/train_cropped_mtcnn'
pos_dir = os.path.join(base_dir, 'yawnCrop')
neg_dir = os.path.join(base_dir, 'no_yawnCrop')


import os

def count_images(folder_path):
    valid_extensions = ('.jpg', '.jpeg', '.png', '.bmp', '.tiff')
    return len([f for f in os.listdir(folder_path) if f.lower().
  ↪endswith(valid_extensions)])

pos_dir = os.path.join(base_dir, 'yawnCrop')
neg_dir = os.path.join(base_dir, 'no_yawnCrop')

num_pos_images = count_images(pos_dir)
num_neg_images = count_images(neg_dir)

print(f"Number of images in yawnCrop: {num_pos_images}")
print(f"Number of images in no_yawnCrop: {num_neg_images}")
```

```
Number of images in yawnCrop: 723
Number of images in no_yawnCrop: 725
```

0.1.2 Use DLIB Face Detector Method 1

```
[ ]: # --- 2. Load dlib models ---
detector = dlib.get_frontal_face_detector()
# Change the path to where you placed the .dat file
predictor_path = '/kaggle/input/dat-data/shape_predictor_68_face_landmarks.dat'
predictor = dlib.shape_predictor(predictor_path)

# --- 3. EAR / MAR functions ---
def eye_aspect_ratio(eye):
    A = dist.euclidean(eye[1], eye[5])
    B = dist.euclidean(eye[2], eye[4])
    C = dist.euclidean(eye[0], eye[3])
    ear = (A + B) / (2.0 * C)
    return ear

def mouth_aspect_ratio(mouth):
    A = dist.euclidean(mouth[2], mouth[10])
    B = dist.euclidean(mouth[4], mouth[8])
    C = dist.euclidean(mouth[0], mouth[6])
    mar = (A + B) / (2.0 * C)
    return mar
```

```
[ ]: # --- 4. Feature extraction ---
def extract_features_from_image(img_path):
    """
    Returns (ear, mar) or None if landmarks / face not found.
    """
    img = cv2.imread(img_path)
    if img is None:
        return None
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = detector(gray, 1)
    if len(faces) == 0:
        return None

    # For simplicity use the first face
    face = faces[0]
    shape = predictor(gray, face)
    shape = face_utils.shape_to_np(shape)  # (68,2)

    # landmarks for eyes
    (l_start, l_end) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]  # (42, 48)
    (r_start, r_end) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
    left_eye = shape[l_start:l_end]
    right_eye = shape[r_start:r_end]
    # landmarks for mouth
    (m_start, m_end) = face_utils.FACIAL_LANDMARKS_IDXS["mouth"]  # (48, 68)
    mouth = shape[m_start:m_end]

    # compute EAR
    left_ear = eye_aspect_ratio(left_eye)
    right_ear = eye_aspect_ratio(right_eye)
    ear = (left_ear + right_ear) / 2.0

    # compute MAR
    mar = mouth_aspect_ratio(mouth)

    return ear, mar
```

```
[ ]: def build_dataset(pos_dir, neg_dir):
         features = []
         labels = []
         # positive (yawn) = 1
         for fname in os.listdir(pos_dir):
             if not fname.lower().endswith(('.jpg','png','jpeg')):
                 continue
             path = os.path.join(pos_dir, fname)
             feat = extract_features_from_image(path)
             if feat is not None:
                 features.append(feat)
                 labels.append(1)

         # negative (no_yawn) = 0
         for fname in os.listdir(neg_dir):
             if not fname.lower().endswith(('.jpg','png','jpeg')):
                 continue
             path = os.path.join(neg_dir, fname)
             feat = extract_features_from_image(path)
             if feat is not None:
                 features.append(feat)
                 labels.append(0)

         return np.array(features), np.array(labels)
```

```
[ ]: # Extract data
     X, y = build_dataset(pos_dir, neg_dir)
     print("Feature array shape:", X.shape, "Labels shape:", y.shape)

     # Some sanity check
     if X.shape[0] == 0:
         raise ValueError("No valid EAR/MAR features found - maybe landmark␣
      ↪detection failed on all images.")
```

```
# lots of images skips due to bad light conditions
```

```
Feature array shape: (421, 2) Labels shape: (421,)
```

[ ]:
```
counts = np.bincount(y)
print("Count of 0s:", counts[0])
print("Count of 1s:", counts[1])
```

```
Count of 0s: 170
Count of 1s: 251
```

[ ]:
```
# --- 5. Train/Test split ---
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

print(X.shape, X_train.shape, X_test.shape)
print(y.shape, y_train.shape, y_test.shape)
```

```
(421, 2) (336, 2) (85, 2)
(421,) (336,) (85,)
```

[ ]:
```
# --- 6. Train a simple model (RandomForest) ---
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
```

[ ]:
```
RandomForestClassifier(random_state=42)
```

[ ]:
```
# --- 7. Evaluate ---
y_pred = clf.predict(X_test)
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.97      0.96        34
           1       0.98      0.96      0.97        51

    accuracy                           0.96        85
   macro avg       0.96      0.97      0.96        85
weighted avg       0.97      0.96      0.96        85

Confusion Matrix:
 [[33  1]
 [ 2 49]]
```

[ ]:

```

```

0.2 USE DLIB Face Detector Method 2

0.2.1 MAR Calculation in previous one is not accurate, This one perfect

[ ]:

```
import dlib
import cv2
import numpy as np

# EAR calculation helper
def eye_aspect_ratio(eye):
    # eye is 6 points (x, y)
    A = np.linalg.norm(eye[1] - eye[5])
    B = np.linalg.norm(eye[2] - eye[4])
    C = np.linalg.norm(eye[0] - eye[3])
    ear = (A + B) / (2.0 * C)
    return ear

# MAR calculation helper
def mouth_aspect_ratio(mouth):
    A = np.linalg.norm(mouth[13] - mouth[19])  # 14-20 in 1-based indexing
    B = np.linalg.norm(mouth[14] - mouth[18])  # 15-19
    C = np.linalg.norm(mouth[15] - mouth[17])  # 16-18
    D = np.linalg.norm(mouth[12] - mouth[16])  # 13-17
    mar = (A + B + C) / (2.0 * D)
    return mar

# Initialize dlib's face detector and landmark predictor globally to avoid␣
 ↪re-loading
detector = dlib.get_frontal_face_detector()
predictor_path = '/kaggle/input/dat-data/shape_predictor_68_face_landmarks.dat'␣
 ↪ # Update this path
predictor = dlib.shape_predictor(predictor_path)

def extract_features_from_image(image_path):
    try:
        img = cv2.imread(image_path)
        if img is None:
            print(f"Warning: Unable to read image {image_path}")
            return None

        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        rects = detector(gray, 1)

        if len(rects) == 0:
            print(f"No face detected in {image_path}")
            return None

        # Use first detected face only
        rect = rects[0]
        shape = predictor(gray, rect)
        shape_np = np.zeros((68, 2), dtype='int')

        for i in range(68):
            shape_np[i] = (shape.part(i).x, shape.part(i).y)

        # Extract eye and mouth landmarks (0-based indexing)
        left_eye = shape_np[42:48]   # points 42-47
        right_eye = shape_np[36:42]  # points 36-41
        mouth = shape_np[48:68]   # points 48-67

        left_ear = eye_aspect_ratio(left_eye)
        right_ear = eye_aspect_ratio(right_eye)
        ear = (left_ear + right_ear) / 2.0
        mar = mouth_aspect_ratio(mouth)

        return [ear, mar]

    except Exception as e:
        print(f"Error processing {image_path}: {e}")
        return None


import os
import numpy as np

def build_dataset(pos_dir, neg_dir):
    features = []
    labels = []
    fail_count = 0

    for folder, label in [(pos_dir, 1), (neg_dir, 0)]:
        for fname in os.listdir(folder):
            if not fname.lower().endswith(('.jpg', '.jpeg', '.png')):
                continue
            path = os.path.join(folder, fname)
            feat = extract_features_from_image(path)
            if feat is not None:
                features.append(feat)
                labels.append(label)
            else:
                fail_count += 1
                print(f"Failed to extract features from {path}")
    print(f"Feature extraction failed on {fail_count} images")
    return np.array(features), np.array(labels)
```

[ ]:
```
# Extract data
X, y = build_dataset(pos_dir, neg_dir)
print("Feature array shape:", X.shape, "Labels shape:", y.shape)

# Some sanity check
if X.shape[0] == 0:
    raise ValueError("No valid EAR/MAR features found - maybe landmark␣
 ↪detection failed on all images.")
```

[ ]:
```
```

[ ]:
```
# See Counts of images


def extract_features_from_image(image_path):
    try:
        img = cv2.imread(image_path)
        if img is None:
            return "unreadable"

        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        rects = detector(gray, 1)
        if len(rects) == 0:
            return "no_face"

        rect = rects[0]
        shape = predictor(gray, rect)
        shape_np = np.zeros((68, 2), dtype='int')
        for i in range(68):
            shape_np[i] = (shape.part(i).x, shape.part(i).y)

        left_eye = shape_np[42:48]
        right_eye = shape_np[36:42]
        mouth = shape_np[48:68]

        left_ear = eye_aspect_ratio(left_eye)
        right_ear = eye_aspect_ratio(right_eye)
        ear = (left_ear + right_ear) / 2.0
        mar = mouth_aspect_ratio(mouth)

        return [ear, mar]

    except Exception:
        return "error"

def build_dataset(pos_dir, neg_dir):
    features = []
    labels = []
    counters = {"no_face": 0, "unreadable": 0, "error": 0, "success": 0}

    for folder, label in [(pos_dir, 1), (neg_dir, 0)]:
        for fname in os.listdir(folder):
            if not fname.lower().endswith(('.jpg', '.jpeg', '.png')):
                continue
            path = os.path.join(folder, fname)
            feat = extract_features_from_image(path)
            if isinstance(feat, list):
                features.append(feat)
                labels.append(label)
                counters["success"] += 1
            else:
                counters[feat] += 1

    print(f"Summary of feature extraction:")
    print(f"  Successful: {counters['success']}")
    print(f"  No face detected: {counters['no_face']}")
    print(f"  Unreadable images: {counters['unreadable']}")
    print(f"  Other errors: {counters['error']}")

    return np.array(features), np.array(labels)
```

[ ]:
```
# Extract data
X, y = build_dataset(pos_dir, neg_dir)
print("Feature array shape:", X.shape, "Labels shape:", y.shape)

# Some sanity check
if X.shape[0] == 0:
    raise ValueError("No valid EAR/MAR features found - maybe landmark␣
 ↪detection failed on all images.")
```

```
Summary of feature extraction:
  Successful: 421
  No face detected: 1027
  Unreadable images: 0
  Other errors: 0
Feature array shape: (421, 2) Labels shape: (421,)
```

[ ]:
```
```

[ ]:
```
```

0.2.2 USE OPEN CV DNN FOR Face Crop Best for all alignment of Images in General

```
[ ]: import cv2
     import dlib
     import numpy as np
     import os

     # EAR helper
     def eye_aspect_ratio(eye):
         A = np.linalg.norm(eye[1] - eye[5])
         B = np.linalg.norm(eye[2] - eye[4])
         C = np.linalg.norm(eye[0] - eye[3])
         return (A + B) / (2.0 * C)

     # MAR helper
     def mouth_aspect_ratio(mouth):
         A = np.linalg.norm(mouth[13] - mouth[19])
         B = np.linalg.norm(mouth[14] - mouth[18])
         C = np.linalg.norm(mouth[15] - mouth[17])
         D = np.linalg.norm(mouth[12] - mouth[16])
         return (A + B + C) / (2.0 * D)

     # Load dlib landmarks predictor
     predictor_path = '/kaggle/input/dat-data/shape_predictor_68_face_landmarks.dat'␣
      ↪ # Update path
     predictor = dlib.shape_predictor(predictor_path)

     # Load OpenCV DNN face detector (deploy prototxt + caffemodel from opencv␣
      ↪github)
     modelFile = "/kaggle/input/open-cv-dnn-files/deploy.prototxt"
     weightsFile = "/kaggle/input/open-cv-dnn-files/res10_300x300_ssd_iter_140000.
      ↪caffemodel"
     net = cv2.dnn.readNetFromCaffe(modelFile, weightsFile)

     def preprocess_image(img, target_width=600):
         h, w = img.shape[:2]
         if w > target_width:
             scale = target_width / w
             img = cv2.resize(img, (target_width, int(h * scale)))
         return img

     def detect_face_opencv_dnn(img):
         h, w = img.shape[:2]
         blob = cv2.dnn.blobFromImage(cv2.resize(img, (300, 300)), 1.0,
                                      (300, 300), (104.0, 177.0, 123.0))
         net.setInput(blob)
```

```
    detections = net.forward()
    max_confidence = 0
    box = None

    # Find the detection with highest confidence > threshold
    for i in range(0, detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence > 0.5 and confidence > max_confidence:
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            max_confidence = confidence

    if box is None:
        return None
    box = box.astype("int")
    (startX, startY, endX, endY) = box
    # Clamp coords to image size
    startX = max(0, startX)
    startY = max(0, startY)
    endX = min(w - 1, endX)
    endY = min(h - 1, endY)
    return (startX, startY, endX, endY)

def extract_features_from_image(image_path):
    try:
        img = cv2.imread(image_path)
        if img is None:
            return "unreadable"

        img = preprocess_image(img, target_width=600)

        face_box = detect_face_opencv_dnn(img)
        if face_box is None:
            return "no_face"

        startX, startY, endX, endY = face_box
        face_img = img[startY:endY, startX:endX]
        gray = cv2.cvtColor(face_img, cv2.COLOR_BGR2GRAY)

        rect = dlib.rectangle(0, 0, face_img.shape[1], face_img.shape[0])
        shape = predictor(gray, rect)

        shape_np = np.zeros((68, 2), dtype='int')
        for i in range(68):
            shape_np[i] = (shape.part(i).x, shape.part(i).y)

        left_eye = shape_np[42:48]
        right_eye = shape_np[36:42]
        mouth = shape_np[48:68]

        left_ear = eye_aspect_ratio(left_eye)
        right_ear = eye_aspect_ratio(right_eye)
        ear = (left_ear + right_ear) / 2.0
        mar = mouth_aspect_ratio(mouth)

        return [ear, mar]

    except Exception as e:
        return "error"

def build_dataset(pos_dir, neg_dir):
    features = []
    labels = []
    counters = {"no_face": 0, "unreadable": 0, "error": 0, "success": 0}

    for folder, label in [(pos_dir, 1), (neg_dir, 0)]:
        for fname in os.listdir(folder):
            if not fname.lower().endswith(('.jpg', '.jpeg', '.png')):
                continue
            path = os.path.join(folder, fname)
            feat = extract_features_from_image(path)
            if isinstance(feat, list):
                features.append(feat)
                labels.append(label)
                counters["success"] += 1
            else:
                counters[feat] += 1

    print(f"Summary of feature extraction:")
    print(f"  Successful: {counters['success']}")
    print(f"  No face detected: {counters['no_face']}")
    print(f"  Unreadable images: {counters['unreadable']}")
    print(f"  Other errors: {counters['error']}")

    return np.array(features), np.array(labels)
```

[ ]:
```
# Extract data
X, y = build_dataset(pos_dir, neg_dir)
print("Feature array shape:", X.shape, "Labels shape:", y.shape)

# Some sanity check
if X.shape[0] == 0:
    raise ValueError("No valid EAR/MAR features found - maybe landmark␣
 ↪detection failed on all images.")
```

```
Summary of feature extraction:
  Successful: 1444
  No face detected: 4
  Unreadable images: 0
  Other errors: 0
Feature array shape: (1444, 2) Labels shape: (1444,)
```

[ ]:
```
counts = np.bincount(y)
print("Count of 0s:", counts[0])
print("Count of 1s:", counts[1])
```

```
Count of 0s: 724
Count of 1s: 720
```

[ ]:
```
# --- Train/Test split ---
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
↪random_state=42)

print(X.shape, X_train.shape, X_test.shape)
print(y.shape, y_train.shape, y_test.shape)
```

```
(1444, 2) (866, 2) (578, 2)
(1444,) (866,) (578,)
```

[ ]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# List of classifiers to try
classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "Support Vector Machine": SVC(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
}

for name, clf in classifiers.items():
    print(f"\n=== {name} ===")
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
=== Logistic Regression ===
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.86      0.79       288
           1       0.83      0.67      0.74       290

    accuracy                           0.77       578
   macro avg       0.78      0.77      0.76       578
weighted avg       0.78      0.77      0.76       578

Confusion Matrix:
 [[248  40]
 [ 95 195]]

=== K-Nearest Neighbors ===
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.76      0.76       288
           1       0.76      0.75      0.76       290

    accuracy                           0.76       578
   macro avg       0.76      0.76      0.76       578
weighted avg       0.76      0.76      0.76       578

Confusion Matrix:
 [[219  69]
 [ 72 218]]

=== Decision Tree ===
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.68      0.68       288
           1       0.68      0.68      0.68       290

    accuracy                           0.68       578
   macro avg       0.68      0.68      0.68       578
weighted avg       0.68      0.68      0.68       578

Confusion Matrix:
 [[196  92]
 [ 92 198]]

=== Random Forest ===
Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.72      0.76      0.74       288
           1       0.75      0.71      0.73       290

    accuracy                           0.74       578
   macro avg       0.74      0.74      0.74       578
weighted avg       0.74      0.74      0.74       578

Confusion Matrix:
 [[219  69]
 [ 84 206]]

=== Support Vector Machine ===
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.82      0.78       288
           1       0.80      0.73      0.76       290

    accuracy                           0.77       578
   macro avg       0.77      0.77      0.77       578
weighted avg       0.77      0.77      0.77       578

Confusion Matrix:
 [[235  53]
 [ 79 211]]

=== Gradient Boosting ===
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.82      0.78       288
           1       0.80      0.73      0.77       290

    accuracy                           0.78       578
   macro avg       0.78      0.78      0.77       578
weighted avg       0.78      0.78      0.77       578

Confusion Matrix:
 [[235  53]
 [ 77 213]]
```

[ ]:
```
from sklearn.model_selection import train_test_split

test_sizes = [0.2, 0.3, 0.4, 0.5]
```

```python
classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "Support Vector Machine": SVC(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
}

for ts in test_sizes:
    print(f"\n\n======= Test size = {ts} =======")
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=ts,␣
 ↪random_state=42)

    print(f"Dataset shapes: X={X.shape}, X_train={X_train.shape},␣
 ↪X_test={X_test.shape}")
    print(f"               y={y.shape}, y_train={y_train.shape}, y_test={y_test.
 ↪shape}")

    for name, clf in classifiers.items():
        print(f"\n--- Classifier: {name} ---")
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        print("Classification Report:\n", classification_report(y_test, y_pred))
        print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
======= Test size = 0.2 =======
Dataset shapes: X=(1444, 2), X_train=(1155, 2), X_test=(289, 2)
               y=(1444,), y_train=(1155,), y_test=(289,)

--- Classifier: Logistic Regression ---
Classification Report:
               precision    recall  f1-score   support

           0       0.74      0.81      0.77       145
           1       0.79      0.71      0.75       144

    accuracy                           0.76       289
   macro avg       0.76      0.76      0.76       289
weighted avg       0.76      0.76      0.76       289

Confusion Matrix:
 [[118  27]
 [ 42 102]]
```

```
--- Classifier: K-Nearest Neighbors ---
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.74      0.75       145
           1       0.74      0.77      0.76       144

    accuracy                           0.75       289
   macro avg       0.75      0.75      0.75       289
weighted avg       0.75      0.75      0.75       289

Confusion Matrix:
 [[107  38]
 [ 33 111]]

--- Classifier: Decision Tree ---
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.65      0.66       145
           1       0.66      0.68      0.67       144

    accuracy                           0.66       289
   macro avg       0.66      0.66      0.66       289
weighted avg       0.66      0.66      0.66       289

Confusion Matrix:
 [[94 51]
 [46 98]]

--- Classifier: Random Forest ---
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.76      0.75       145
           1       0.75      0.73      0.74       144

    accuracy                           0.74       289
   macro avg       0.74      0.74      0.74       289
weighted avg       0.74      0.74      0.74       289

Confusion Matrix:
 [[110  35]
 [ 39 105]]

--- Classifier: Support Vector Machine ---
Classification Report:
              precision    recall  f1-score   support
           0       0.76      0.77      0.76       145
           1       0.77      0.75      0.76       144

    accuracy                           0.76       289
   macro avg       0.76      0.76      0.76       289
weighted avg       0.76      0.76      0.76       289

Confusion Matrix:
 [[112  33]
 [ 36 108]]

--- Classifier: Gradient Boosting ---
Classification Report:
               precision    recall  f1-score   support

           0       0.75      0.79      0.77       145
           1       0.78      0.74      0.76       144

    accuracy                           0.76       289
   macro avg       0.77      0.76      0.76       289
weighted avg       0.77      0.76      0.76       289

Confusion Matrix:
 [[115  30]
 [ 38 106]]


======= Test size = 0.3 =======
Dataset shapes: X=(1444, 2), X_train=(1010, 2), X_test=(434, 2)
               y=(1444,), y_train=(1010,), y_test=(434,)

--- Classifier: Logistic Regression ---
Classification Report:
               precision    recall  f1-score   support

           0       0.74      0.84      0.78       213
           1       0.82      0.71      0.76       221

    accuracy                           0.77       434
   macro avg       0.78      0.78      0.77       434
weighted avg       0.78      0.77      0.77       434

Confusion Matrix:
 [[178  35]
 [ 63 158]]

--- Classifier: K-Nearest Neighbors ---
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.77      0.77       213
           1       0.78      0.78      0.78       221

    accuracy                           0.77       434
   macro avg       0.77      0.77      0.77       434
weighted avg       0.77      0.77      0.77       434

Confusion Matrix:
 [[164  49]
 [ 49 172]]

--- Classifier: Decision Tree ---
Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.70      0.70       213
           1       0.71      0.71      0.71       221

    accuracy                           0.71       434
   macro avg       0.70      0.70      0.70       434
weighted avg       0.71      0.71      0.71       434

Confusion Matrix:
 [[149  64]
 [ 64 157]]

--- Classifier: Random Forest ---
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.77      0.76       213
           1       0.78      0.76      0.77       221

    accuracy                           0.76       434
   macro avg       0.77      0.77      0.76       434
weighted avg       0.77      0.76      0.76       434

Confusion Matrix:
 [[165  48]
 [ 54 167]]

--- Classifier: Support Vector Machine ---
Classification Report:
              precision    recall  f1-score   support
           0       0.75      0.79      0.77       213
           1       0.79      0.75      0.77       221

    accuracy                           0.77       434
   macro avg       0.77      0.77      0.77       434
weighted avg       0.77      0.77      0.77       434

Confusion Matrix:
 [[169  44]
 [ 55 166]]

--- Classifier: Gradient Boosting ---
Classification Report:
               precision    recall  f1-score   support

           0       0.75      0.77      0.76       213
           1       0.77      0.75      0.76       221

    accuracy                           0.76       434
   macro avg       0.76      0.76      0.76       434
weighted avg       0.76      0.76      0.76       434

Confusion Matrix:
 [[164  49]
 [ 55 166]]


======= Test size = 0.4 =======
Dataset shapes: X=(1444, 2), X_train=(866, 2), X_test=(578, 2)
               y=(1444,), y_train=(866,), y_test=(578,)

--- Classifier: Logistic Regression ---
Classification Report:
               precision    recall  f1-score   support

           0       0.73      0.86      0.79       288
           1       0.84      0.68      0.75       290

    accuracy                           0.77       578
   macro avg       0.78      0.77      0.77       578
weighted avg       0.78      0.77      0.77       578

Confusion Matrix:
 [[249  39]
 [ 92 198]]

--- Classifier: K-Nearest Neighbors ---
Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.76      0.78      0.77       288
           1       0.78      0.76      0.77       290

    accuracy                           0.77       578
   macro avg       0.77      0.77      0.77       578
weighted avg       0.77      0.77      0.77       578

Confusion Matrix:
 [[226  62]
 [ 70 220]]

--- Classifier: Decision Tree ---
Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.70      0.70       288
           1       0.70      0.70      0.70       290

    accuracy                           0.70       578
   macro avg       0.70      0.70      0.70       578
weighted avg       0.70      0.70      0.70       578

Confusion Matrix:
 [[202  86]
 [ 86 204]]

--- Classifier: Random Forest ---
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.80      0.78       288
           1       0.79      0.75      0.77       290

    accuracy                           0.77       578
   macro avg       0.77      0.77      0.77       578
weighted avg       0.77      0.77      0.77       578

Confusion Matrix:
 [[229  59]
 [ 73 217]]

--- Classifier: Support Vector Machine ---
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.82      0.78       288
           1       0.80      0.72      0.76       290

    accuracy                           0.77       578
   macro avg       0.77      0.77      0.77       578
weighted avg       0.77      0.77      0.77       578

Confusion Matrix:
 [[235  53]
 [ 80 210]]

--- Classifier: Gradient Boosting ---
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.78      0.77       288
           1       0.78      0.75      0.76       290

    accuracy                           0.77       578
   macro avg       0.77      0.77      0.77       578
weighted avg       0.77      0.77      0.77       578

Confusion Matrix:
 [[226  62]
 [ 72 218]]


======= Test size = 0.5 =======
Dataset shapes: X=(1444, 2), X_train=(722, 2), X_test=(722, 2)
               y=(1444,), y_train=(722,), y_test=(722,)

--- Classifier: Logistic Regression ---
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.86      0.78       355
           1       0.83      0.68      0.75       367

    accuracy                           0.77       722
   macro avg       0.78      0.77      0.77       722
weighted avg       0.78      0.77      0.77       722

Confusion Matrix:
 [[305  50]
 [118 249]]

--- Classifier: K-Nearest Neighbors ---
Classification Report:
              precision    recall  f1-score   support
           0       0.75      0.80      0.77       355
           1       0.79      0.74      0.77       367

    accuracy                           0.77       722
   macro avg       0.77      0.77      0.77       722
weighted avg       0.77      0.77      0.77       722

Confusion Matrix:
 [[283  72]
 [ 95 272]]

--- Classifier: Decision Tree ---
Classification Report:
               precision    recall  f1-score   support

           0       0.68      0.73      0.71       355
           1       0.72      0.67      0.70       367

    accuracy                           0.70       722
   macro avg       0.70      0.70      0.70       722
weighted avg       0.70      0.70      0.70       722

Confusion Matrix:
 [[259  96]
 [120 247]]

--- Classifier: Random Forest ---
Classification Report:
               precision    recall  f1-score   support

           0       0.70      0.79      0.74       355
           1       0.77      0.68      0.72       367

    accuracy                           0.73       722
   macro avg       0.74      0.73      0.73       722
weighted avg       0.74      0.73      0.73       722

Confusion Matrix:
 [[280  75]
 [118 249]]

--- Classifier: Support Vector Machine ---
Classification Report:
               precision    recall  f1-score   support

           0       0.73      0.81      0.77       355
           1       0.80      0.72      0.75       367
    accuracy                           0.76       722
   macro avg       0.77      0.76      0.76       722
weighted avg       0.77      0.76      0.76       722

Confusion Matrix:
 [[288  67]
 [104 263]]

--- Classifier: Gradient Boosting ---
Classification Report:
               precision    recall  f1-score   support

           0       0.71      0.82      0.76       355
           1       0.80      0.68      0.73       367

    accuracy                           0.75       722
   macro avg       0.75      0.75      0.75       722
weighted avg       0.76      0.75      0.75       722

Confusion Matrix:
 [[291  64]
 [117 250]]
```

```
[ ]:
```

0.2.3 MORE Features Than EAR and MAR

```
[ ]: def extract_features_from_image(image_path):
         try:
             img = cv2.imread(image_path)
             if img is None:
                 return "unreadable"

             img = preprocess_image(img, target_width=600)

             face_box = detect_face_opencv_dnn(img)
             if face_box is None:
                 return "no_face"

             startX, startY, endX, endY = face_box
             face_img = img[startY:endY, startX:endX]
             gray = cv2.cvtColor(face_img, cv2.COLOR_BGR2GRAY)

             rect = dlib.rectangle(0, 0, face_img.shape[1], face_img.shape[0])
             shape = predictor(gray, rect)
```

```
        shape_np = np.zeros((68, 2), dtype='double')  # use float for solvePnP
        for i in range(68):
            shape_np[i] = (shape.part(i).x, shape.part(i).y)

        left_eye = shape_np[42:48]
        right_eye = shape_np[36:42]
        mouth = shape_np[48:68]

        # EAR and MAR
        left_ear = eye_aspect_ratio(left_eye)
        right_ear = eye_aspect_ratio(right_eye)
        ear = (left_ear + right_ear) / 2.0
        mar = mouth_aspect_ratio(mouth)

        ### NEW FEATURE 1: Head Pose Estimation ###
        # 3D model points of standard landmarks
        model_points = np.array([
            (0.0, 0.0, 0.0),             # Nose tip
            (0.0, -330.0, -65.0),        # Chin
            (-225.0, 170.0, -135.0),     # Left eye left corner
            (225.0, 170.0, -135.0),      # Right eye right corner
            (-150.0, -150.0, -125.0),    # Left mouth corner
            (150.0, -150.0, -125.0)      # Right mouth corner
        ])

        image_points = np.array([
            shape_np[30],  # Nose tip
            shape_np[8],   # Chin
            shape_np[36],  # Left eye left corner
            shape_np[45],  # Right eye right corner
            shape_np[48],  # Left mouth corner
            shape_np[54]   # Right mouth corner
        ], dtype='double')

        focal_length = face_img.shape[1]
        center = (face_img.shape[1] / 2, face_img.shape[0] / 2)
        camera_matrix = np.array(
            [[focal_length, 0, center[0]],
             [0, focal_length, center[1]],
             [0, 0, 1]], dtype="double"
        )
        dist_coeffs = np.zeros((4, 1))  # Assuming no lens distortion

        success, rotation_vector, _ = cv2.solvePnP(
            model_points, image_points, camera_matrix, dist_coeffs, flags=cv2.
↪SOLVEPNP_ITERATIVE)
```

```
        pitch, yaw, roll = cv2.Rodrigues(rotation_vector)[0][:, 0]  #␣
↪Simplified rotation approx

        ### NEW FEATURE 2: Mouth Width Ratio ###
        mouth_width = np.linalg.norm(shape_np[54] - shape_np[48])
        face_width = face_img.shape[1]
        mouth_width_ratio = mouth_width / face_width

        ### NEW FEATURE 3: Eye Distance Ratio ###
        eye_distance = np.linalg.norm(shape_np[42] - shape_np[39])  # outer␣
↪corners
        eye_distance_ratio = eye_distance / face_width

        ### Final feature vector ###
        return [ear, mar, pitch, yaw, roll, mouth_width_ratio,␣
↪eye_distance_ratio]

    except Exception as e:
        return "error"
```

```
[ ]: def build_dataset(pos_dir, neg_dir):
         features = []
         labels = []
         counters = {"no_face": 0, "unreadable": 0, "error": 0, "success": 0}

         for folder, label in [(pos_dir, 1), (neg_dir, 0)]:
             for fname in os.listdir(folder):
                 if not fname.lower().endswith(('.jpg', '.jpeg', '.png')):
                     continue
                 path = os.path.join(folder, fname)
                 feat = extract_features_from_image(path)
                 if isinstance(feat, list):
                     features.append(feat)
                     labels.append(label)
                     counters["success"] += 1
                 else:
                     counters[feat] += 1

         print(f"Summary of feature extraction:")
         print(f"  Successful: {counters['success']}")
         print(f"  No face detected: {counters['no_face']}")
         print(f"  Unreadable images: {counters['unreadable']}")
         print(f"  Other errors: {counters['error']}")

         return np.array(features), np.array(labels)
```

```
[ ]: # Extract data
     X, y = build_dataset(pos_dir, neg_dir)
     print("Feature array shape:", X.shape, "Labels shape:", y.shape)

     # Some sanity check
     if X.shape[0] == 0:
         raise ValueError("No valid EAR/MAR features found - maybe landmark␣
      ↪detection failed on all images.")
```

```
Summary of feature extraction:
  Successful: 1444
  No face detected: 4
  Unreadable images: 0
  Other errors: 0
Feature array shape: (1444, 7) Labels shape: (1444,)
```

```
[ ]: counts = np.bincount(y)
     print("Count of 0s:", counts[0])
     print("Count of 1s:", counts[1])
```

```
Count of 0s: 724
Count of 1s: 720
```

```
[ ]: # --- Train/Test split ---
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
      ↪random_state=42)

     print(X.shape, X_train.shape, X_test.shape)
     print(y.shape, y_train.shape, y_test.shape)
```

```
(1444, 7) (866, 7) (578, 7)
(1444,) (866,) (578,)
```

```
[ ]: from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
     from sklearn.linear_model import LogisticRegression
     from sklearn.svm import SVC
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.metrics import classification_report, confusion_matrix
     import numpy as np

     # List of classifiers to try
     classifiers = {
         "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
         "K-Nearest Neighbors": KNeighborsClassifier(),
         "Decision Tree": DecisionTreeClassifier(random_state=42),
         "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
```

```python
    "Support Vector Machine": SVC(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
}

for name, clf in classifiers.items():
    print(f"\n=== {name} ===")
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
=== Logistic Regression ===
Classification Report:
               precision    recall  f1-score   support

           0       0.73      0.86      0.79       288
           1       0.83      0.68      0.75       290

    accuracy                           0.77       578
   macro avg       0.78      0.77      0.77       578
weighted avg       0.78      0.77      0.77       578

Confusion Matrix:
 [[249  39]
 [ 93 197]]

=== K-Nearest Neighbors ===
Classification Report:
               precision    recall  f1-score   support

           0       0.76      0.80      0.77       288
           1       0.79      0.74      0.76       290

    accuracy                           0.77       578
   macro avg       0.77      0.77      0.77       578
weighted avg       0.77      0.77      0.77       578

Confusion Matrix:
 [[229  59]
 [ 74 216]]

=== Decision Tree ===
Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.81      0.79       288
           1       0.80      0.76      0.78       290

    accuracy                           0.78       578
   macro avg       0.78      0.78      0.78       578
weighted avg       0.78      0.78      0.78       578

Confusion Matrix:
 [[234  54]
 [ 71 219]]

=== Random Forest ===
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.83      0.80       288
           1       0.82      0.76      0.79       290

    accuracy                           0.79       578
   macro avg       0.80      0.79      0.79       578
weighted avg       0.80      0.79      0.79       578

Confusion Matrix:
 [[238  50]
 [ 69 221]]

=== Support Vector Machine ===
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.83      0.78       288
           1       0.81      0.70      0.75       290

    accuracy                           0.77       578
   macro avg       0.77      0.77      0.77       578
weighted avg       0.77      0.77      0.77       578

Confusion Matrix:
 [[240  48]
 [ 86 204]]

=== Gradient Boosting ===
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.82      0.79       288
           1       0.81      0.76      0.78       290

    accuracy                           0.79       578
   macro avg       0.79      0.79      0.79       578
weighted avg       0.79      0.79      0.79       578

Confusion Matrix:
 [[236  52]
 [ 71 219]]
```

[ ]:
```python
from sklearn.ensemble import VotingClassifier

voting_clf = VotingClassifier(
    estimators=[
        ('lr', LogisticRegression()),
        ('rf', RandomForestClassifier()),
        ('gb', GradientBoostingClassifier())
    ],
    voting='soft'  # use predicted probabilities
)

voting_clf.fit(X_train, y_train)
```

[ ]:
```
VotingClassifier(estimators=[('lr', LogisticRegression()),
                             ('rf', RandomForestClassifier()),
                             ('gb', GradientBoostingClassifier())],
                 voting='soft')
```

[ ]:
```python
y_pred = voting_clf.predict(X_test)
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.78      0.85      0.81       288
           1       0.84      0.76      0.79       290

    accuracy                           0.80       578
   macro avg       0.81      0.80      0.80       578
weighted avg       0.81      0.80      0.80       578

Confusion Matrix:
 [[245  43]
 [ 71 219]]
```

[ ]:
```
!pip install keras-facenet --quiet
```

```
  Preparing metadata (setup.py) … done
                         1.9/1.9 MB
32.6 MB/s eta 0:00:00a 0:00:01
```

```
                              1.3/1.3 MB
52.8 MB/s eta 0:00:00
  Building wheel for keras-facenet (setup.py) … done
```

[ ]:

```

```

0.2.4 Use Face Embeddings with the features Extracted

[ ]:

```python
from keras.models import load_model
from keras_facenet import FaceNet
import cv2
import numpy as np

# Load the model
embedder = FaceNet()
```

```
I0000 00:00:1759684140.291201      36 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 13942 MB memory:  -> device:
0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5
I0000 00:00:1759684140.292053      36 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:1 with 13942 MB memory:  -> device:
1, name: Tesla T4, pci bus id: 0000:00:05.0, compute capability: 7.5
```

[ ]:

```python
def get_face_embedding(img, face_box):
    startX, startY, endX, endY = face_box
    face = img[startY:endY, startX:endX]
    if face.size == 0:
        return None
    face = cv2.resize(face, (160, 160))
    face = face.astype('float32') / 255.0
    face = np.expand_dims(face, axis=0)
    embedding = embedder.embeddings(face)
    return embedding.flatten()


def extract_features_from_image(image_path):
    try:
        img = cv2.imread(image_path)
        if img is None:
            return "unreadable"

        img = preprocess_image(img, target_width=600)
        face_box = detect_face_opencv_dnn(img)
        if face_box is None:
            return "no_face"

        startX, startY, endX, endY = face_box
        face_img = img[startY:endY, startX:endX]
        gray = cv2.cvtColor(face_img, cv2.COLOR_BGR2GRAY)

        rect = dlib.rectangle(0, 0, face_img.shape[1], face_img.shape[0])
        shape = predictor(gray, rect)

        shape_np = np.zeros((68, 2), dtype='double')
        for i in range(68):
            shape_np[i] = (shape.part(i).x, shape.part(i).y)

        # EAR and MAR
        left_eye = shape_np[42:48]
        right_eye = shape_np[36:42]
        mouth = shape_np[48:68]

        left_ear = eye_aspect_ratio(left_eye)
        right_ear = eye_aspect_ratio(right_eye)
        ear = (left_ear + right_ear) / 2.0
        mar = mouth_aspect_ratio(mouth)

        # Head Pose
        model_points = np.array([
            (0.0, 0.0, 0.0),
            (0.0, -330.0, -65.0),
            (-225.0, 170.0, -135.0),
            (225.0, 170.0, -135.0),
            (-150.0, -150.0, -125.0),
            (150.0, -150.0, -125.0)
        ])
        image_points = np.array([
            shape_np[30],
            shape_np[8],
            shape_np[36],
            shape_np[45],
            shape_np[48],
            shape_np[54]
        ], dtype='double')

        focal_length = face_img.shape[1]
        center = (face_img.shape[1] / 2, face_img.shape[0] / 2)
        camera_matrix = np.array(
            [[focal_length, 0, center[0]],
             [0, focal_length, center[1]],
             [0, 0, 1]], dtype="double"
        )
        dist_coeffs = np.zeros((4, 1))
        success, rotation_vector, _ = cv2.solvePnP(
            model_points, image_points, camera_matrix, dist_coeffs, flags=cv2.
↪SOLVEPNP_ITERATIVE)

        pitch, yaw, roll = cv2.Rodrigues(rotation_vector)[0][:, 0]

        # Extra ratios
        mouth_width = np.linalg.norm(shape_np[54] - shape_np[48])
        eye_distance = np.linalg.norm(shape_np[42] - shape_np[39])
        face_width = face_img.shape[1]
        mouth_ratio = mouth_width / face_width
        eye_ratio = eye_distance / face_width

        # Face Embedding
        embedding = get_face_embedding(img, face_box)
        if embedding is None:
            return "embedding_error"

        # Final feature vector
        return np.concatenate([[ear, mar, pitch, yaw, roll, mouth_ratio,␣
↪eye_ratio], embedding])

    except Exception as e:
        return "error"



def build_dataset(pos_dir, neg_dir):
    features = []
    labels = []
    counters = {
        "success": 0,
        "no_face": 0,
        "unreadable": 0,
        "embedding_error": 0,
        "error": 0
    }

    for folder, label in [(pos_dir, 1), (neg_dir, 0)]:
        for fname in os.listdir(folder):
            if not fname.lower().endswith(('.jpg', '.jpeg', '.png')):
                continue

            path = os.path.join(folder, fname)
            feat = extract_features_from_image(path)

            if isinstance(feat, (list, np.ndarray)):
                features.append(feat)
                labels.append(label)
                counters["success"] += 1
            elif isinstance(feat, str) and feat in counters:
                counters[feat] += 1
            else:
                print(f"Unknown error or return type for {fname}: {feat}")
                counters["error"] += 1

    print("\n Summary of feature extraction:")
    for k, v in counters.items():
        print(f"  {k}: {v}")

    return np.array(features), np.array(labels)
```

[ ]:
```
# Extract data
X, y = build_dataset(pos_dir, neg_dir)
print("Feature array shape:", X.shape, "Labels shape:", y.shape)
```

[ ]:
```
# Some sanity check
if X.shape[0] == 0:
    raise ValueError("No valid EAR/MAR features found - maybe landmark␣
 ↪detection failed on all images.")


counts = np.bincount(y)
print("Count of 0s:", counts[0])
print("Count of 1s:", counts[1])
```

```
Count of 0s: 724
Count of 1s: 720
```

[ ]:
```
# --- Train/Test split ---
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=42)

print(X.shape, X_train.shape, X_test.shape)
print(y.shape, y_train.shape, y_test.shape)
```

```
(1444, 519) (866, 519) (578, 519)
(1444,) (866,) (578,)
```

[ ]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```python
import numpy as np

# List of classifiers to try
classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "Support Vector Machine": SVC(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
}

for name, clf in classifiers.items():
    print(f"\n=== {name} ===")
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
=== Logistic Regression ===
Classification Report:
               precision    recall  f1-score   support

           0       0.76      0.84      0.80       288
           1       0.83      0.73      0.78       290

    accuracy                           0.79       578
   macro avg       0.79      0.79      0.79       578
weighted avg       0.79      0.79      0.79       578

Confusion Matrix:
 [[243  45]
 [ 77 213]]

=== K-Nearest Neighbors ===
Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.80      0.79       288
           1       0.80      0.79      0.79       290

    accuracy                           0.79       578
   macro avg       0.79      0.79      0.79       578
weighted avg       0.79      0.79      0.79       578

Confusion Matrix:
 [[229  59]
 [ 61 229]]

=== Decision Tree ===
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.73      0.74       288
           1       0.74      0.76      0.75       290

    accuracy                           0.75       578
   macro avg       0.75      0.75      0.75       578
weighted avg       0.75      0.75      0.75       578

Confusion Matrix:
 [[211  77]
 [ 69 221]]

=== Random Forest ===
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.83      0.81       288
           1       0.82      0.79      0.81       290

    accuracy                           0.81       578
   macro avg       0.81      0.81      0.81       578
weighted avg       0.81      0.81      0.81       578

Confusion Matrix:
 [[239  49]
 [ 60 230]]

=== Support Vector Machine ===
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.84      0.79       288
           1       0.82      0.72      0.76       290

    accuracy                           0.78       578
   macro avg       0.78      0.78      0.78       578
weighted avg       0.78      0.78      0.78       578

Confusion Matrix:
 [[242  46]
 [ 82 208]]
```

```
=== Gradient Boosting ===
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.81      0.81       288
           1       0.81      0.81      0.81       290

    accuracy                           0.81       578
   macro avg       0.81      0.81      0.81       578
weighted avg       0.81      0.81      0.81       578

Confusion Matrix:
 [[233  55]
 [ 54 236]]
```

[ ]:

```

```

0.2.5 Select top 10 to 15 Features from embeddings

[ ]:

```python
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Feature selection
k = 15  # you can change to 10-15
selector = SelectKBest(score_func=f_classif, k=k)
X_selected = selector.fit_transform(X_scaled, y)

# Get selected feature indices
selected_indices = selector.get_support(indices=True)
print("Selected feature indices:", selected_indices)
```

```
Selected feature indices: [  0   1   2   3   4   5   6 511 512 513 514 515 516
517 518]

/usr/local/lib/python3.11/dist-
packages/sklearn/feature_selection/_univariate_selection.py:112: UserWarning:
Features [  7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23
24
  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42
  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78
  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96
  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114
 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132
 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150
 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168
 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186
 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204
 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222
 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258
 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276
 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294
 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312
 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330
 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348
 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366
 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384
 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402
 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420
 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438
 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456
 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474
 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492
 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510
 511 512 513 514 515 516 517 518] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx,
UserWarning)
/usr/local/lib/python3.11/dist-
packages/sklearn/feature_selection/_univariate_selection.py:113: RuntimeWarning:
invalid value encountered in divide
  f = msb / msw
```

[ ]:
```
# --- Train/Test split ---
X_train, X_test, y_train, y_test = train_test_split(X_selected , y, test_size=0.
 ↪4, random_state=42)

print(X.shape, X_train.shape, X_test.shape)
print(y.shape, y_train.shape, y_test.shape)
```

```
(1444, 519) (866, 15) (578, 15)
(1444,) (866,) (578,)
```

[ ]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
```

```python
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# List of classifiers to try
classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "Support Vector Machine": SVC(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
}

for name, clf in classifiers.items():
    print(f"\n=== {name} ===")
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
=== Logistic Regression ===
Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.84      0.81       288
           1       0.83      0.76      0.79       290

    accuracy                           0.80       578
   macro avg       0.80      0.80      0.80       578
weighted avg       0.80      0.80      0.80       578

Confusion Matrix:
 [[242  46]
 [ 71 219]]

=== K-Nearest Neighbors ===
Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.83      0.81       288
           1       0.82      0.79      0.80       290

    accuracy                           0.81       578
   macro avg       0.81      0.81      0.81       578
weighted avg       0.81      0.81      0.81       578
```

```
Confusion Matrix:
 [[238  50]
 [ 62 228]]

=== Decision Tree ===
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.74      0.75       288
           1       0.75      0.77      0.76       290

    accuracy                           0.75       578
   macro avg       0.75      0.75      0.75       578
weighted avg       0.75      0.75      0.75       578

Confusion Matrix:
 [[214  74]
 [ 68 222]]

=== Random Forest ===
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.83      0.82       288
           1       0.83      0.81      0.82       290

    accuracy                           0.82       578
   macro avg       0.82      0.82      0.82       578
weighted avg       0.82      0.82      0.82       578

Confusion Matrix:
 [[239  49]
 [ 56 234]]

=== Support Vector Machine ===
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.84      0.81       288
           1       0.83      0.76      0.79       290

    accuracy                           0.80       578
   macro avg       0.80      0.80      0.80       578
weighted avg       0.80      0.80      0.80       578

Confusion Matrix:
 [[243  45]
 [ 69 221]]
```

```
=== Gradient Boosting ===
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.81      0.81       288
           1       0.81      0.81      0.81       290

    accuracy                           0.81       578
   macro avg       0.81      0.81      0.81       578
weighted avg       0.81      0.81      0.81       578

Confusion Matrix:
 [[233  55]
 [ 54 236]]
```

```
[ ]: from sklearn.ensemble import VotingClassifier

     voting_clf = VotingClassifier(
         estimators=[
             ('lr', LogisticRegression()),
             ('rf', RandomForestClassifier()),
             ('gb', GradientBoostingClassifier())
         ],
         voting='soft'  # use predicted probabilities
     )

     voting_clf.fit(X_train, y_train)
```

```
[ ]: VotingClassifier(estimators=[('lr', LogisticRegression()),
                                  ('rf', RandomForestClassifier()),
                                  ('gb', GradientBoostingClassifier())],
                      voting='soft')
```

```
[ ]: y_pred = voting_clf.predict(X_test)
     print("Classification Report:\n", classification_report(y_test, y_pred))
     print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.82      0.81       288
           1       0.82      0.79      0.81       290

    accuracy                           0.81       578
   macro avg       0.81      0.81      0.81       578
weighted avg       0.81      0.81      0.81       578
```