```
#Step 1: Dataset Summary & Visualization
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import os

# Define paths
base_path = "/content/Dataset/Dataset"
train_dir = os.path.join(base_path, "train")
val_dir = os.path.join(base_path, "val")

# Image size & batch size
img_size = (150, 150)
batch_size = 32

# Data Generators
train_gen = ImageDataGenerator(rescale=1./255, zoom_range=0.2,
horizontal_flip=True)
val_gen = ImageDataGenerator(rescale=1./255)

train_data = train_gen.flow_from_directory(
    train_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary'
)

val_data = val_gen.flow_from_directory(
    val_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary'
)

# Count image samples
def count_images(path):
    counts = {}
    total = 0
    for label in os.listdir(path):
        label_path = os.path.join(path, label)
        count = len(os.listdir(label_path))
        counts[label] = count
        total += count
    return counts, total

train_counts, train_total = count_images(train_dir)
val_counts, val_total = count_images(val_dir)

print("📊 Dataset Summary:")
print(f"Train Set: {train_total} images {train_counts}")
print(f"Validation Set: {val_total} images {val_counts}")

# 📈 Visualization
labels = list(train_counts.keys())
train_vals = [train_counts[label] for label in labels]
val_vals = [val_counts[label] for label in labels]

x_pos = range(len(labels))
bar_width = 0.35

fig, ax = plt.subplots()
ax.bar(x_pos, train_vals, width=bar_width, label='Train', color='blue')
ax.bar([p + bar_width for p in x_pos], val_vals, width=bar_width,
label='Validation', color='red')

ax.set_xticks([p + bar_width / 2 for p in x_pos])
ax.set_xticklabels(labels)
ax.set_ylabel("Number of Images")
ax.set_title("Dataset Distribution by Class")
ax.legend()
plt.show()
#Step 2: CNN Model Summary
from tensorflow.keras import layers, models

# Define CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150,
3)),
    layers.MaxPooling2D(2, 2),
```

```python
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')  # Binary classification (open vs closed)
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Display model summary
model.summary()

#Step 3: Epoch Status with Accuracy & Loss (Training + Visualization)
import matplotlib.pyplot as plt

# Train the model
history = model.fit(
    train_data,
    epochs=10,
    validation_data=val_data
)

# 📊 Plot Accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title("Training vs Validation Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

# 📉 Plot Loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title("Training vs Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
#Step 4: Save the Trained Model
import os

# Define path to save model
save_dir = "/content/drive/MyDrive/Drowsiness_Model"
model_path = os.path.join(save_dir, "eye_status_model.keras")

# Create folder if it doesn't exist
os.makedirs(save_dir, exist_ok=True)

# Save the model
model.save(model_path)
print(f"✅ Model saved at: {model_path}")
# Step 5: EVALUATE THE MODEL PERFORMANCE
val_loss, val_acc = model.evaluate(val_data)
print(f"✅ Validation Accuracy: {val_acc * 100:.2f}%")
# Step 6: Evaluation on Test Set
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Path to test data
test_dir = '/content/Dataset/Dataset/test'  # change path if needed

# Create a test data generator (only rescale)
test_gen = ImageDataGenerator(rescale=1./255)

# Load test images
test_data = test_gen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    shuffle=False  # Important: don’t shuffle for evaluation
)
```

```
# Evaluate the model on test set
test_loss, test_acc = model.evaluate(test_data)
print(f"✅ Test Accuracy: {test_acc * 100:.2f}%")
#Step 6: Generate Classification Report & Confusion Matrix
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Get predictions from model on test data
y_pred_probs = model.predict(test_data)  # probabilities
y_pred = (y_pred_probs > 0.5).astype("int32").flatten()  # convert to
class labels

# Step 2: Get true labels
y_true = test_data.classes

# Step 3: Generate Classification Report
print("📊 Classification Report:")
print(classification_report(y_true, y_pred,
target_names=test_data.class_indices.keys()))

# Step 4: Confusion Matrix
cm = confusion_matrix(y_true, y_pred)

# Visualize Confusion Matrix
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=test_data.class_indices.keys(),
yticklabels=test_data.class_indices.keys())
plt.title("🧾 Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

#Step 7: Sample Image Predictions with Labels
import matplotlib.pyplot as plt
import numpy as np

# Get one batch of images and labels from test_data
```

```
test_data.reset()  # Reset if generator was previously used
images, labels = next(test_data)  # Get a single batch

# Predict using the model
preds = model.predict(images)
pred_labels = (preds > 0.5).astype("int32").flatten()

# Map class index to class name
class_names = list(test_data.class_indices.keys())

# Plot first 8 images with actual and predicted labels
plt.figure(figsize=(16, 8))
for i in range(8):
    plt.subplot(2, 4, i+1)
    plt.imshow(images[i])
    plt.axis('off')
    actual = class_names[int(labels[i])]
    predicted = class_names[int(pred_labels[i])]
    color = 'green' if actual == predicted else 'red'
    plt.title(f"Actual: {actual}\nPred: {predicted}", color=color)

plt.suptitle("🔍 Sample Predictions from Test Set", fontsize=16)
plt.tight_layout()
plt.show()
import cv2
import dlib
import numpy as np
import tensorflow as tf
from imutils import face_utils
import pygame
import time
from scipy.spatial import distance

# Load CNN Model
model = tf.keras.models.load_model("eye_status_model.keras")

# Load dlib's face detector and shape predictor
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
```

```
# Initialize Pygame for sound alarm
pygame.mixer.init()
pygame.mixer.music.load(r"C:\Users\Ferina\Desktop\Project\alarm.wav")

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

# EAR threshold
EAR_THRESHOLD = 0.25

# Start video capture
cap = cv2.VideoCapture(0)

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

    # Convert to grayscale for EAR calculation
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    rects = detector(gray)

    for rect in rects:
        shape = predictor(gray, rect)
        shape = face_utils.shape_to_np(shape)

        # Get eye coordinates
        left_eye = shape[36:42]
        right_eye = shape[42:48]

        # Compute EAR
        leftEAR = eye_aspect_ratio(left_eye)
        rightEAR = eye_aspect_ratio(right_eye)
        avg_ear = (leftEAR + rightEAR) / 2.0
        # Draw landmarks on eyes
        for (x, y) in np.concatenate((left_eye, right_eye), axis=0):
            cv2.circle(frame, (x, y), 2, (0, 255, 0), -1)

        # Extract eye region for CNN
        (x1, y1) = np.min(np.vstack((left_eye, right_eye)), axis=0)
        (x2, y2) = np.max(np.vstack((left_eye, right_eye)), axis=0)
        margin = 5
        eye_frame = frame[y1 - margin:y2 + margin, x1 - margin:x2 +
margin]

        try:
            eye_frame = cv2.resize(eye_frame, (150, 150)) / 255.0
            eye_frame = eye_frame.reshape(1, 150, 150, 3)  # RGB input
            prediction = model.predict(eye_frame, verbose=0)
            eye_status = "Open" if prediction[0][0] > 0.5 else "Closed"
        except:
            eye_status = "Unknown"

        # Combine logic for drowsiness
        if avg_ear < EAR_THRESHOLD and eye_status == "Closed":
            color = (0, 0, 255)
            status_text = "Status: Closed"
            if not pygame.mixer.music.get_busy():
                pygame.mixer.music.play()
        else:
            color = (255, 255, 255)
            status_text = "Status: Open"
            pygame.mixer.music.stop()

        # Display EAR and Status
        cv2.putText(frame, f"EAR: {avg_ear:.2f}", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 2)
        cv2.putText(frame, status_text, (180, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 2)

    # Show frame
    cv2.imshow("Drowsiness Detection", frame)

    # Exit key
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

# Cleanup
cap.release()
cv2.destroyAllWindows()
pygame.mixer.quit()
```