```
# Install the necessary libraries
!pip install --upgrade pip  # Upgrade pip for latest features
!pip install -q kaggle  # Install Kaggle API
```

```
Requirement already satisfied: pip in /usr/local/lib/python3.11/dist-packages (2
Collecting pip
  Downloading pip-25.0.1-py3-none-any.whl.metadata (3.7 kB)
Downloading pip-25.0.1-py3-none-any.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 23.0 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 24.1.2
    Uninstalling pip-24.1.2:
      Successfully uninstalled pip-24.1.2
Successfully installed pip-25.0.1
```

```
# Kaggle API for installing dataset
from google.colab import files
import os

# 1. Upload kaggle.json (if you haven't already)
if not os.path.exists("/root/.kaggle/kaggle.json"):
    print("Upload your kaggle.json file")
    files.upload() # Select your kaggle.json file

    # 2. Move kaggle.json to the required directory
    !mkdir -p ~/.kaggle
    !cp kaggle.json ~/.kaggle/
    !chmod 600 ~/.kaggle/kaggle.json
    print("kaggle.json configured.")
else:
    print("kaggle.json already exists.")

# 3. Download and unzip the specific dataset
# Using the command you provided:
!kaggle datasets download prasadvpatil/mrl-dataset -p /content/dataset --unzip
print("Dataset downloaded and unzipped to /content/dataset")

# 4. Check the structure (Important!)
# List the contents to understand the folder structure
print("\nContents of /content/dataset:")
!ls /content/dataset
```

```
Upload your kaggle.json file
Choose Files kaggle.json
```

- **kaggle.json**(application/json) - 62 bytes, last modified: 4/19/2025 - 100% done

```
Saving kaggle.json to kaggle.json
kaggle.json configured.
Dataset URL: https://www.kaggle.com/datasets/prasadvpatil/mrl-dataset
License(s): CC0-1.0
Dataset downloaded and unzipped to /content/dataset

Contents of /content/dataset:
train
```

```
# PyTorch libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms, datasets, models
from torchvision.models import ResNet18_Weights # Or other weights if using a differ

# Data handling and visualization
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import time
import copy

# OpenCV and Dlib for real-time processing
import cv2
import dlib
from google.colab.patches import cv2_imshow # Special function for showing images in

# For plotting and metrics
from sklearn.metrics import confusion_matrix
import seaborn as sns

# For saving notebook to GitHub
from google.colab import drive # Needed if saving via Drive intermediate step

print(f"PyTorch version: {torch.__version__}")
print(f"Dlib version: {dlib.__version__}")
print(f"OpenCV version: {cv2.__version__}")

# Check for GPU availability
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

```
PyTorch version: 2.6.0+cu124
Dlib version: 19.24.6
OpenCV version: 4.11.0
Using device: cuda
```

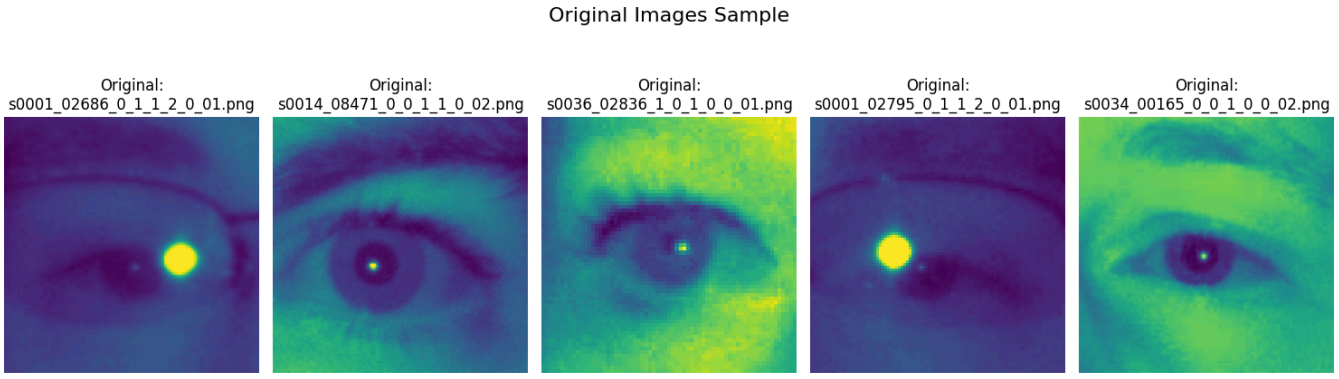
```
import os
import shutil
import random
from sklearn.model_selection import train_test_split

def split_dataset(source_dir_open_eyes, source_dir_closed_eyes, output_dir, test_siz
    """Splits data into train, validation, and test sets.

    Args:
        source_dir_open_eyes: Path to the 'yawn' directory.
        source_dir_closed_eyes: Path to the 'no_yawn' directory.
        output_dir: The directory where the split datasets will be saved.
        test_size: Proportion of data to include in the test split.
        val_size: Proportion of data to include in the validation split.
    """

    # Create output directories if they don't exist
    for split in ["train", "val", "test"]:
        for label in ["Open_Eyes", "Closed_Eyes"]:
            os.makedirs(os.path.join(output_dir, split, label), exist_ok=True)

    # Splitting the yawn data
    open_eyes_files = [f for f in os.listdir(source_dir_open_eyes) if os.path.isfile
    open_eyes_train, open_eyes_temp = train_test_split(open_eyes_files, test_size=te
    open_eyes_val, open_eyes_test = train_test_split(open_eyes_temp, test_size=test_

    # Splitting the no_yawn data
    closed_eyes_files = [f for f in os.listdir(source_dir_closed_eyes) if os.path.is
    closed_eyes_train, closed_eyes_temp = train_test_split(closed_eyes_files, test_s
    closed_eyes_val, closed_eyes_test = train_test_split(closed_eyes_temp, test_size

    # Copy files to respective directories
    def copy_files(files, source_dir, dest_dir):
        for file in files:
            shutil.copy(os.path.join(source_dir, file), dest_dir)

    copy_files(open_eyes_train, source_dir_open_eyes, os.path.join(output_dir, "trai
    copy_files(open_eyes_val, source_dir_open_eyes, os.path.join(output_dir, "val",
    copy_files(open_eyes_test, source_dir_open_eyes, os.path.join(output_dir, "test"

    copy_files(closed_eyes_train, source_dir_closed_eyes, os.path.join(output_dir, "
    copy_files(closed_eyes_val, source_dir_closed_eyes, os.path.join(output_dir, "va
    copy_files(closed_eyes_test, source_dir_closed_eyes, os.path.join(output_dir, "t

# Example usage:
source_open_eyes = "/content/dataset/Open_Eyes"
source_closed_eyes = "/content/dataset/Closed_Eyes"
output_dataset_dir = "/content/split_dataset"
split_dataset(source_open_eyes, source_closed_eyes, output_dataset_dir)
```

```
# Function to visualize images from a directory
def visualize_images(directory, num_images=5):
    # Find image files (checking for common extensions)
    extensions = ('*.jpg', '*.jpeg', '*.png', '*.bmp', '*.gif')
    images = []
    for ext in extensions:
        images.extend([f for f in os.listdir(directory) if f.lower().endswith(ext.re

    if not images:
        print(f"No images found in {directory}")
        return

    num_images = min(num_images, len(images)) # Ensure we don't try to display more

    plt.figure(figsize=(15, 5))
    for i in range(num_images):
        img_path = os.path.join(directory, images[i])
        try:
            img = Image.open(img_path)
            plt.subplot(1, num_images, i + 1)
            plt.imshow(img)
            plt.title(f"Original:\n{images[i]}")
            plt.axis('off')
        except Exception as e:
            print(f"Error opening/displaying image {img_path}: {e}")
    plt.suptitle("Original Images Sample", fontsize=16)
    plt.tight_layout(rect=[0, 0.03, 1, 0.95]) # Adjust layout to prevent title overl
    plt.show()


def preprocess_image(image_path):
    """
    Preprocesses an image: resizes, converts to grayscale (3 channels), normalizes.

    Args:
        image_path: The path to the image file.

    Returns:
        A preprocessed image tensor ready for a ResNet18 model or None on error.
    """
    try:
        img = Image.open(image_path).convert("RGB") # Ensure RGB
        preprocess = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.Grayscale(num_output_channels=3), # Convert to grayscale with
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.22
        ])
        img_tensor = preprocess(img)
        return img_tensor
```

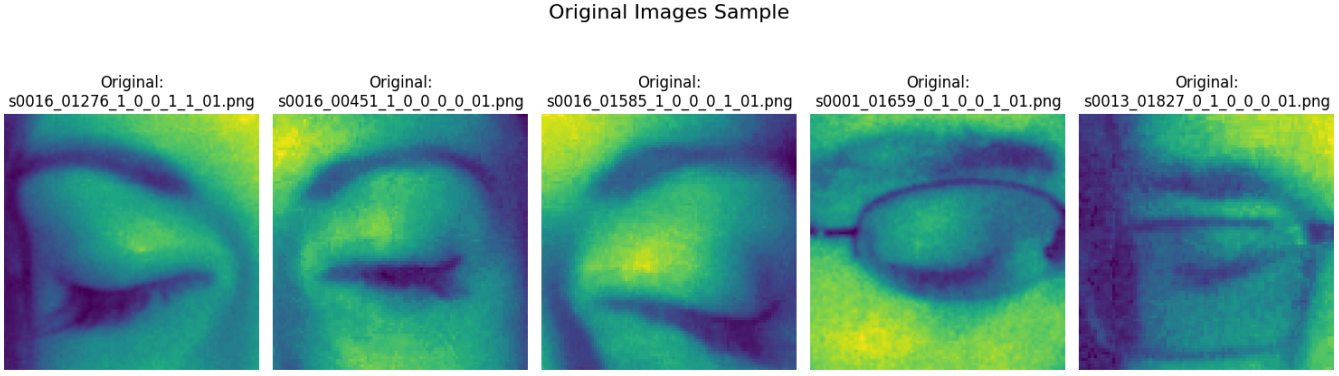
```
        except Exception as e:
            print(f"Error processing image {image_path}: {e}")
            return None # or raise the exception

    # --- Visualization setup ---
    # Define directories (adjust if your path differs)
    # Make sure this path exists and contains yawn images
    # open_eyes_dir = "/content/split_dataset/train/yawn"
    open_eyes_dir = "/content/split_dataset/train/Open_Eyes" # Corrected based on previo
    closed_eyes_dir = "/content/split_dataset/train/Closed_Eyes" # Corrected based on pr


    # Visualize original yawn images
    print(f"Visualizing original images from: {open_eyes_dir}")
    visualize_images(open_eyes_dir)

    # Visualize original no_yawn images
    print(f"\nVisualizing original images from: {closed_eyes_dir}")
    visualize_images(closed_eyes_dir)


    # --- New code to preprocess and display one example ---

    # 1. Get a sample image path (use the first image found in open_eyes_dir)
    print("\nPreprocessing and displaying one example...")
    try:
        extensions = ('*.jpg', '*.jpeg', '*.png', '*.bmp', '*.gif')
        example_image_name = None
        for ext in extensions:
             potential_files = [f for f in os.listdir(open_eyes_dir) if f.lower().endswi
             if potential_files:
                 example_image_name = potential_files[0]
                 break

        if not example_image_name:
            raise FileNotFoundError(f"No image files found in {open_eyes_dir}")

        example_image_path = os.path.join(open_eyes_dir, example_image_name)
        print(f"Selected example image: {example_image_path}")

        # 2. Call preprocess_image
        preprocessed_tensor = preprocess_image(example_image_path)

        if preprocessed_tensor is not None:
            # 3. Prepare tensor for display (denormalize and rearrange)
            # Convert tensor to numpy array
            img_np = preprocessed_tensor.numpy()

            # Transpose dimensions from (C, H, W) to (H, W, C)
            img_np = np.transpose(img_np, (1, 2, 0))
```

```
            # Denormalize the image
            mean = np.array([0.485, 0.456, 0.406])
            std = np.array([0.229, 0.224, 0.225])
            img_np = std * img_np + mean # Apply inverse transform: std * img + mean

            # Clip values to be between 0 and 1
            img_np = np.clip(img_np, 0, 1)

            # 4. Display using plt.imshow
            plt.figure(figsize=(6, 6))
            plt.imshow(img_np)
            # Since it's converted to grayscale with 3 identical channels, it will look
            plt.title(f"Preprocessed Example:\n{example_image_name}\n(Grayscale, Resized
            plt.axis('off')
            plt.show()
        else:
            print("Preprocessing failed for the example image.")

    except FileNotFoundError as fnf_error:
        print(f"Error finding example image: {fnf_error}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

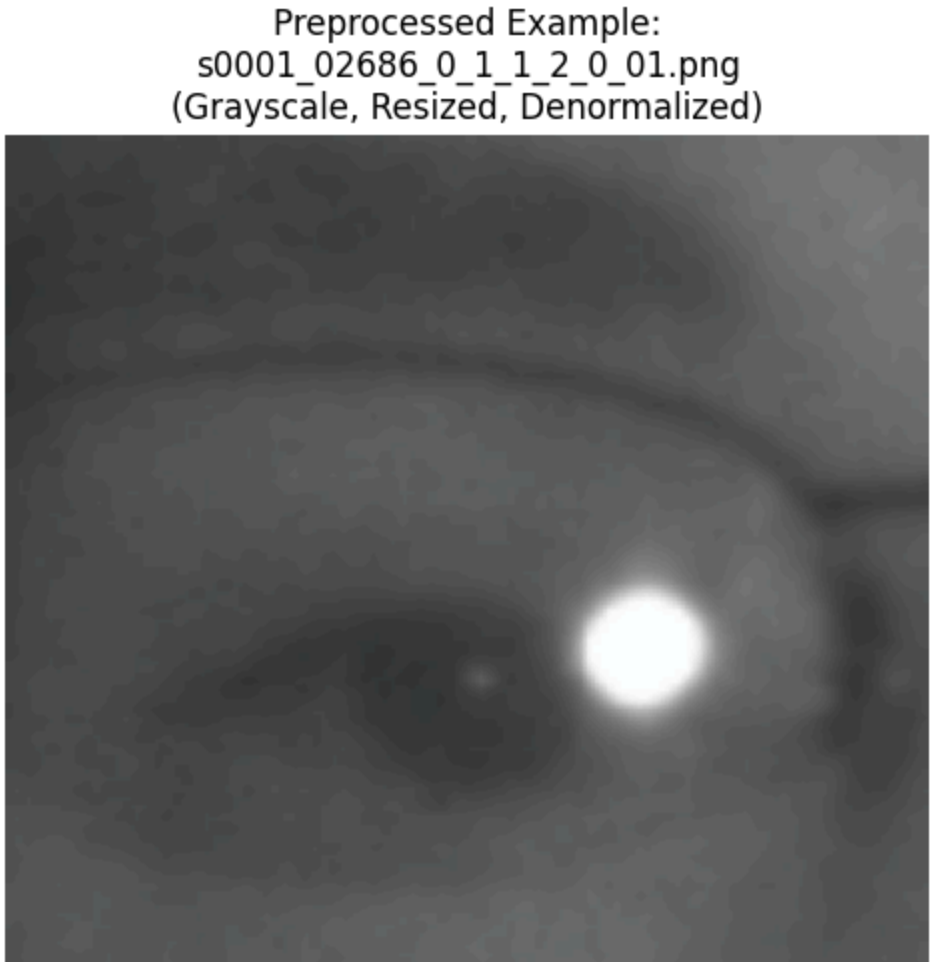
```
Visualizing original images from: /content/split_dataset/train/Open_Eyes
```




```
Visualizing original images from: /content/split_dataset/train/Closed_Eyes
```




```
Preprocessing and displaying one example...
Selected example image: /content/split_dataset/train/Open_Eyes/s0001_02686_0_1_1
```



```
# Define the custom dataset class
# Define the custom dataset class
class YawnDataset(Dataset):
    def __init__(self, data_dir, transform=None):
        self.data_dir = data_dir
        self.transform = transform
        self.image_paths = []
        self.labels = []
        # Add the classes attribute
        self.classes = ["Open_Eyes", "Closed_Eyes"] # Assuming "no_yawn" is 0 and "y

        for class_name in os.listdir(data_dir):
            class_dir = os.path.join(data_dir, class_name)
            if os.path.isdir(class_dir):
                label = 1 if class_name == "yawn" else 0
                for image_name in os.listdir(class_dir):
                    image_path = os.path.join(class_dir, image_name)
                    self.image_paths.append(image_path)
                    self.labels.append(label)

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image_path = self.image_paths[idx]
        image = Image.open(image_path).convert('RGB')
        label = self.labels[idx]

        if self.transform:
            image = self.transform(image)

        return image, label
```

```
# Data Transformations
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'test': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}

# Create datasets
image_datasets = {x: YawnDataset(os.path.join("/content/split_dataset", x), data_tra
dataloaders = {x: DataLoader(image_datasets[x], batch_size=32, shuffle=True, num_wor
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val', 'test']}
class_names = image_datasets['train'].classes


# Load pre-trained ResNet18 model
model = models.resnet18(weights=ResNet18_Weights.DEFAULT)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)  # 2 output classes (yawn, no_yawn)
model = model.to(device)

# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

# Training function
def train_model(model, criterion, optimizer, num_epochs=25):
    since = time.time()
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print('-' * 10)

        for phase in ['train', 'val']:
            if phase == 'train':
```

```
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s'
    print(f'Best val Acc: {best_acc:4f}')

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model

# Train the model
model_trained = train_model(model, criterion, optimizer, num_epochs=10) # Reduced ep

# Testing
def test_model(model, criterion):
    model.eval()  # Set the model to evaluation mode
```

```
        running_loss = 0.0
        running_corrects = 0
        y_true = []
        y_pred = []

        with torch.no_grad():
          for inputs, labels in dataloaders['test']:
              inputs = inputs.to(device)
              labels = labels.to(device)
              outputs = model(inputs)
              _, preds = torch.max(outputs, 1)
              loss = criterion(outputs, labels)

              running_loss += loss.item() * inputs.size(0)
              running_corrects += torch.sum(preds == labels.data)
              y_true.extend(labels.cpu().numpy())
              y_pred.extend(preds.cpu().numpy())

        test_loss = running_loss / dataset_sizes['test']
        test_acc = running_corrects.double() / dataset_sizes['test']
        print(f'Test Loss: {test_loss:.4f} Acc: {test_acc:.4f}')

        cm = confusion_matrix(y_true, y_pred)
        plt.figure(figsize=(8,6))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, ytic
        plt.xlabel("Predicted")
        plt.ylabel("True")
        plt.title("Confusion Matrix")
        plt.show()

        return test_acc

    # Test the model
    test_accuracy = test_model(model_trained, criterion)
```