```
# Install the necessary libraries
!pip install --upgrade pip  # Upgrade pip for latest features
!pip install --pre dlib  # Install pre-built dlib if available
!pip install opencv-python-headless # Force reinstall opencv-python
!pip install -q kaggle  # Install Kaggle API
```

```
Requirement already satisfied: pip in /usr/local/lib/python3.11/dist-packages (24.1.2)
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 32.5 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
Requirement already satisfied: dlib in /usr/local/lib/python3.11/dist-packages (19.24.6)
Requirement already satisfied: opencv-python-headless in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from opencv-python-headless) (2.0.2
```

```
# 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 davidvazquezcic/yawn-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) - 69 bytes, last modified: 4/19/2025 - 100% done
Saving kaggle.json to kaggle.json
kaggle.json configured.
Dataset URL: https://www.kaggle.com/datasets/davidvazquezcic/yawn-dataset
License(s): CC-BY-NC-SA-4.0
Dataset downloaded and unzipped to /content/dataset

Contents of /content/dataset:
'no yawn'   yawn
```

```
# 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 different model

# 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 Colab

# 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}")
```

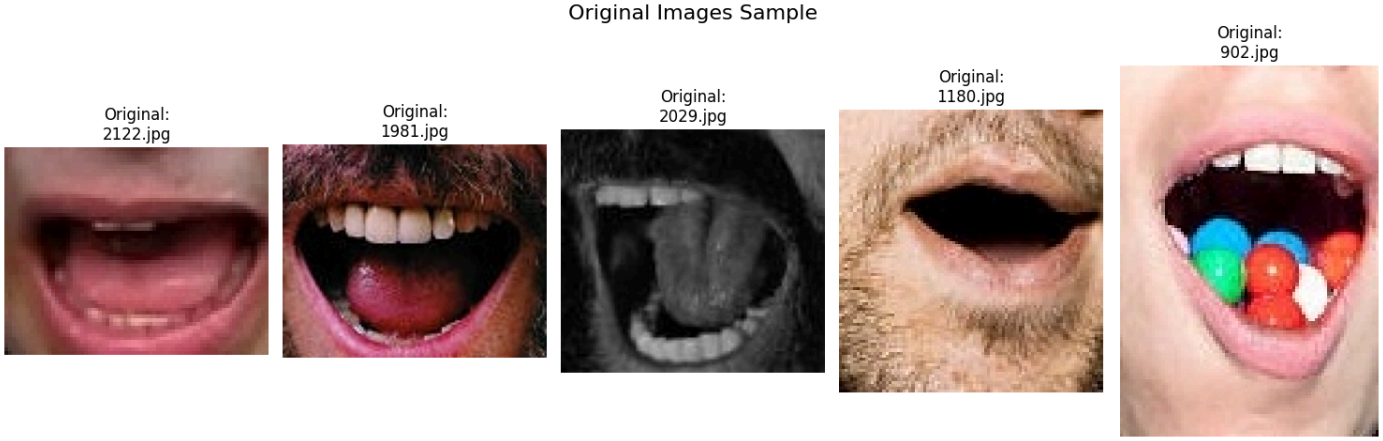
```
PyTorch version: 2.6.0+cu124
Dlib version: 19.24.6
OpenCV version: 4.11.0
Using device: cuda
```

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

def split_dataset(source_dir_yawn, source_dir_no_yawn, output_dir, test_size=0.2, val_size=0.1):
    """Splits data into train, validation, and test sets.

    Args:
        source_dir_yawn: Path to the 'yawn' directory.
        source_dir_no_yawn: 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 ["yawn", "no_yawn"]:
            os.makedirs(os.path.join(output_dir, split, label), exist_ok=True)

    # Splitting the yawn data
    yawn_files = [f for f in os.listdir(source_dir_yawn) if os.path.isfile(os.path.join(source_dir_yawn, f))]
    yawn_train, yawn_temp = train_test_split(yawn_files, test_size=test_size + val_size, random_state=42)
    yawn_val, yawn_test = train_test_split(yawn_temp, test_size=test_size / (test_size + val_size), random_state=42)

    # Splitting the no_yawn data
    no_yawn_files = [f for f in os.listdir(source_dir_no_yawn) if os.path.isfile(os.path.join(source_dir_no_yawn, f))]
    no_yawn_train, no_yawn_temp = train_test_split(no_yawn_files, test_size=test_size + val_size, random_state=42)
    no_yawn_val, no_yawn_test = train_test_split(no_yawn_temp, test_size=test_size / (test_size + val_size), random_state=42)

    # 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(yawn_train, source_dir_yawn, os.path.join(output_dir, "train", "yawn"))
    copy_files(yawn_val, source_dir_yawn, os.path.join(output_dir, "val", "yawn"))
    copy_files(yawn_test, source_dir_yawn, os.path.join(output_dir, "test", "yawn"))

    copy_files(no_yawn_train, source_dir_no_yawn, os.path.join(output_dir, "train", "no_yawn"))
    copy_files(no_yawn_val, source_dir_no_yawn, os.path.join(output_dir, "val", "no_yawn"))
    copy_files(no_yawn_test, source_dir_no_yawn, os.path.join(output_dir, "test", "no_yawn"))

# Example usage:
source_yawn = "/content/dataset/yawn"
source_no_yawn = "/content/dataset/no_yawn"
output_dataset_dir = "/content/split_dataset"
split_dataset(source_yawn, source_no_yawn, output_dataset_dir)



# Function to visualize images from a directory
def visualize_images(directory, num_images=5):
```

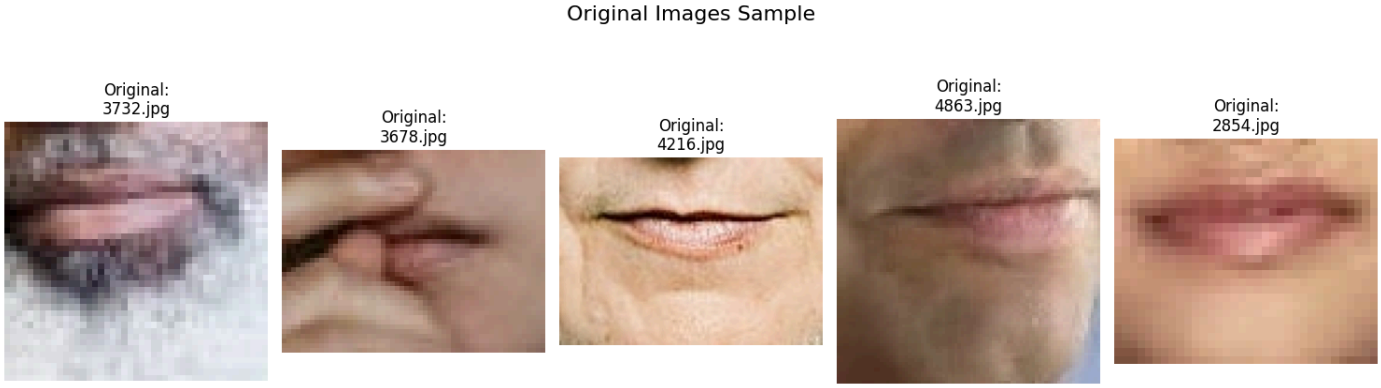
```
    def visualize_images(directory, num_images=5):
        # Find image files (checking for common extensions)
        extensions = ('*.jpg', '*.jpeg', '*.png', '*.bmp', '*.gif')
        images = []
        for ext in extensions:
            images.extend([f for f in os.listdir(directory) if f.lower().endswith(ext.replace('*','')) and os.path.isfile(os.path.joi

        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 images than exist

        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 overlap
        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 3 channels
                transforms.ToTensor(),
                transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]), # Normalize
            ])
            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
    # yawn_dir = "/content/split_dataset/train/yawn"
    yawn_dir = "/content/split_dataset/train/yawn" # Corrected based on previous steps
    no_yawn_dir = "/content/split_dataset/train/no_yawn" # Corrected based on previous steps


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

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


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

    # 1. Get a sample image path (use the first image found in yawn_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(yawn_dir) if f.lower().endswith(ext.replace('*','')) and os.path.isfile(os.path
            if potential_files:
                example_image_name = potential_files[0]
                break

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

    example_image_path = os.path.join(yawn_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 grayscale
        plt.title(f"Preprocessed Example:\n{example_image_name}\n(Grayscale, Resized, Denormalized)")
        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}")
```

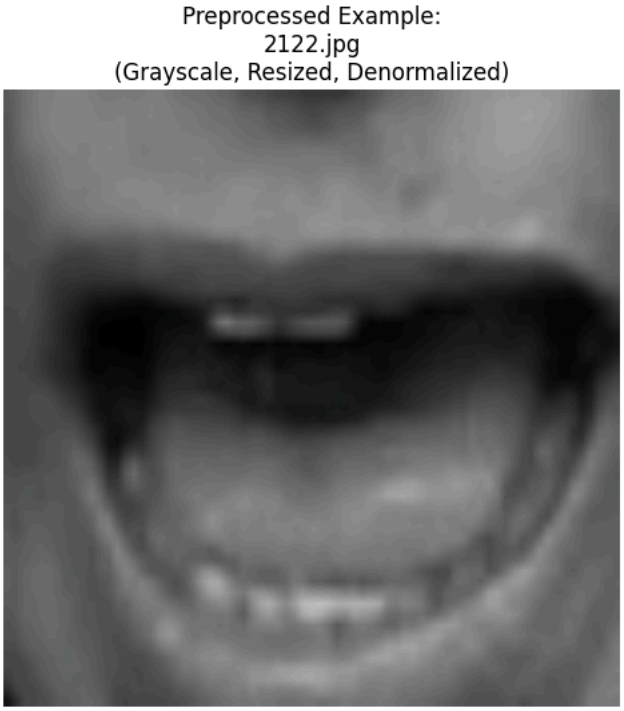
```
Visualizing original images from: /content/split_dataset/train/yawn
```




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




```
Preprocessing and displaying one example...
Selected example image: /content/split_dataset/train/yawn/2122.jpg
```



```
# 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 = ["no_yawn", "yawn"] # Assuming "no_yawn" is 0 and "yawn" is 1

        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_transforms[x]) for x in ['train', 'val', 'test']
dataloaders = {x: DataLoader(image_datasets[x], batch_size=32, shuffle=True, num_workers=4) for x in ['train', 'val', 'test']}
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 epochs for demonstration

    # 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, yticklabels=class_names)
    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)
```