eye_model_pytorch

April 19, 2025

[27]:
```
# === Core Libraries ===
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt

# === PyTorch Libraries ===
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader

# === Scikit-learn ===
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

[28]:
```
IMG_SIZE = 64

def load_data(data_dir):
    X = []
    y = []
    for label, folder in enumerate(['Closed_Eyes', 'Open_Eyes']):
        path = os.path.join(data_dir, folder)
        for img_name in os.listdir(path):
            img_path = os.path.join(path, img_name)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
            X.append(img)
            y.append(label)
    return np.array(X), np.array(y)

# Load and preprocess
X, y = load_data('dataset/train')
X = X / 255.0
X = X.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[29]:
```
class EyeDataset(Dataset):
    def __init__(self, images, labels):
        self.images = torch.tensor(images, dtype=torch.float32)
        self.labels = torch.tensor(labels, dtype=torch.float32)

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

    def __getitem__(self, idx):
        return self.images[idx].permute(2, 0, 1), self.labels[idx]

train_data = EyeDataset(X_train, y_train)
val_data = EyeDataset(X_val, y_val)

train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
val_loader = DataLoader(val_data, batch_size=32)
```

[30]:
```
class EyeCNN(nn.Module):
    def __init__(self):
        super(EyeCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.3)
        self.fc1 = nn.Linear(64 * 14 * 14, 64)
        self.fc2 = nn.Linear(64, 1)

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 14 * 14)
        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        return torch.sigmoid(self.fc2(x)).squeeze()
```

[31]:
```
model = EyeCNN()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
for epoch in range(10):
    model.train()
    total_loss = 0
```

```
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print(f"Epoch {epoch+1} - Loss: {total_loss/len(train_loader):.4f}")
```

```
Epoch 1 - Loss: 0.2366
Epoch 2 - Loss: 0.0573
Epoch 3 - Loss: 0.0428
Epoch 4 - Loss: 0.0189
Epoch 5 - Loss: 0.0204
Epoch 6 - Loss: 0.0155
Epoch 7 - Loss: 0.0098
Epoch 8 - Loss: 0.0042
Epoch 9 - Loss: 0.0045
Epoch 10 - Loss: 0.0042
```

[32]:
```
# save model version ~ 1
torch.save(model.state_dict(), "eye_state_model.pth")
print("  Model saved as eye_state_model.pth")
```

```
 Model saved as eye_state_model.pth
```

[33]:
```
# fine tune model using newly collected dataset
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split

IMG_SIZE = 64

def load_custom_eye_data(path):
    X = []
    y = []
    for label, folder in enumerate(['closed', 'open']):
        folder_path = os.path.join(path, folder)
        for img_name in os.listdir(folder_path):
            img_path = os.path.join(folder_path, img_name)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            if img is None:
                continue
            img = cv2.equalizeHist(img)
            img = cv2.resize(img, (IMG_SIZE, IMG_SIZE)) / 255.0
            X.append(img)
```

```
            y.append(label)
    return np.array(X), np.array(y)

# Load data
X_custom, y_custom = load_custom_eye_data('my_eye_crops')
X_custom = X_custom.reshape(-1, IMG_SIZE, IMG_SIZE, 1)

# Train/val split
X_train, X_val, y_train, y_val = train_test_split(X_custom, y_custom,␣
 ↪test_size=0.2, random_state=42)
```

[34]:
```
# create pytorch dataset
import torch
from torch.utils.data import Dataset, DataLoader

class EyeDataset(Dataset):
    def __init__(self, images, labels):
        self.images = torch.tensor(images, dtype=torch.float32)
        self.labels = torch.tensor(labels, dtype=torch.float32)

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

    def __getitem__(self, idx):
        return self.images[idx].permute(2, 0, 1), self.labels[idx]

train_ds = EyeDataset(X_train, y_train)
val_ds = EyeDataset(X_val, y_val)

train_loader = DataLoader(train_ds, batch_size=16, shuffle=True)
val_loader = DataLoader(val_ds, batch_size=16)
```

[35]:
```
# Reuse PyTorch Model Architecture
import torch.nn as nn
import torch.nn.functional as F

class EyeCNN(nn.Module):
    def __init__(self):
        super(EyeCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.3)
        self.fc1 = nn.Linear(64 * 14 * 14, 64)
        self.fc2 = nn.Linear(64, 1)
```

```
    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 14 * 14)
        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        return torch.sigmoid(self.fc2(x)).squeeze()
```

[36]:
```
# fine tune model
model = EyeCNN()
model.load_state_dict(torch.load("eye_state_model.pth")) # load base model

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0005)

# Fine-tuning loop
for epoch in range(10):
    model.train()
    train_loss = 0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

    model.eval()
    val_loss = 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            val_loss += criterion(outputs, labels).item()

    print(f"Epoch {epoch+1}: Train Loss = {train_loss/len(train_loader):.4f},␣
↪Val Loss = {val_loss/len(val_loader):.4f}")
```

```
Epoch 1: Train Loss = 6.7198, Val Loss = 0.4636
Epoch 2: Train Loss = 0.8381, Val Loss = 0.3120
Epoch 3: Train Loss = 0.3679, Val Loss = 0.1329
Epoch 4: Train Loss = 0.2054, Val Loss = 0.1085
Epoch 5: Train Loss = 0.1653, Val Loss = 0.0883
Epoch 6: Train Loss = 0.1518, Val Loss = 0.0789
```

```
Epoch 7: Train Loss = 0.1333, Val Loss = 0.0677
Epoch 8: Train Loss = 0.1363, Val Loss = 0.0696
Epoch 9: Train Loss = 0.1099, Val Loss = 0.0678
Epoch 10: Train Loss = 0.1083, Val Loss = 0.0733
```

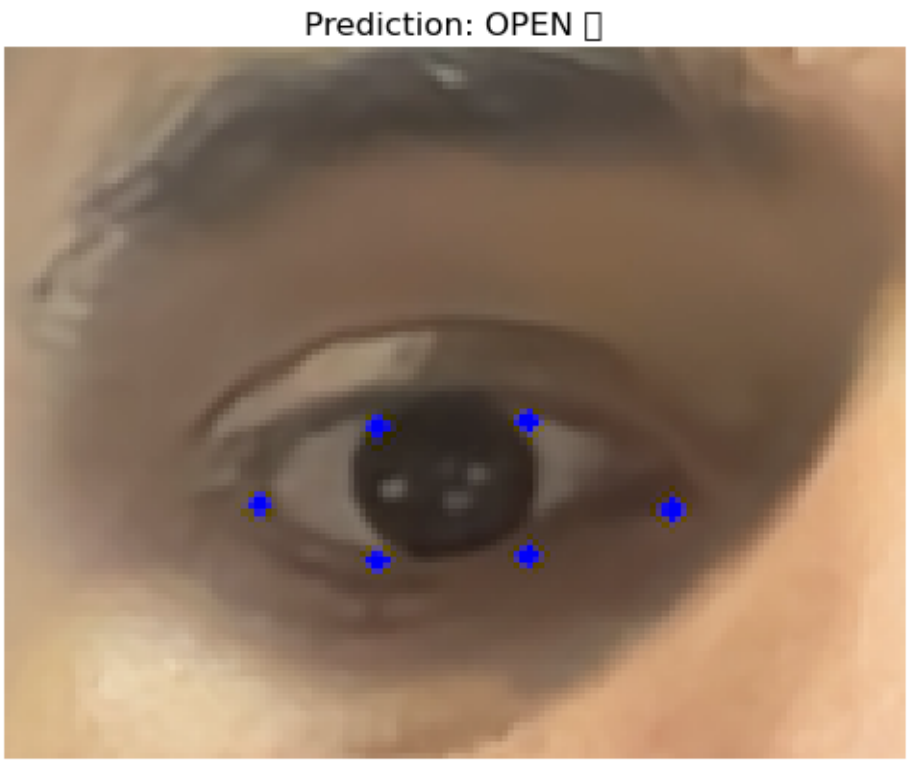
[37]:
```
# save model
torch.save(model.state_dict(), "eye_state_model_finetuned.pth")
print("  Fine-tuned model saved.")
```

```
 Fine-tuned model saved.
```

[39]:
```
# manually test model
import torch
import torch.nn as nn
import cv2
import numpy as np
import matplotlib.pyplot as plt
from torchvision import transforms

# === Define the model (same as trained) ===
class EyeCNN(nn.Module):
    def __init__(self):
        super(EyeCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.3)
        self.fc1 = nn.Linear(64 * 14 * 14, 64)
        self.fc2 = nn.Linear(64, 1)

    def forward(self, x):
        x = self.pool1(torch.relu(self.conv1(x)))
        x = self.pool2(torch.relu(self.conv2(x)))
        x = x.view(-1, 64 * 14 * 14)
        x = self.dropout(x)
        x = torch.relu(self.fc1(x))
        return torch.sigmoid(self.fc2(x)).squeeze()

# === Load model ===
model = EyeCNN()
model.load_state_dict(torch.load("eye_state_model_finetuned.pth",␣
 ↪map_location=torch.device("cpu")))
model.eval()

# === Image transformation ===
transform = transforms.Compose([
    transforms.ToPILImage(),
```

```
    transforms.Grayscale(),
    transforms.Resize((64, 64)),
    transforms.ToTensor()
])

# === Prediction function ===
def predict_eye_state(image_path):
    img = cv2.imread(image_path)
    if img is None:
        print(" Could not read image.")
        return

    preprocessed = transform(img).unsqueeze(0)  # shape: [1, 1, 64, 64]

    with torch.no_grad():
        pred = model(preprocessed).item()

    print(f" Raw prediction value: {pred:.2f}")
    label = "OPEN  " if pred > 0.5 else "CLOSED  "
    print(f" Prediction: Eye is {label}")

    # Show the image
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title(f"Prediction: {label}")
    plt.axis(False)
    plt.show()

# === Test the model with a sample image ===
test_image_path = 'my_eye_crops/cam/open.png'
predict_eye_state(test_image_path)
```

```
 Raw prediction value: 0.69
 Prediction: Eye is OPEN
```



[41]:
```
# live camera test

import cv2
import dlib
import numpy as np
import torch
from torchvision import transforms

# === Load model definition ===
class EyeCNN(nn.Module):
    def __init__(self):
        super(EyeCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.3)
        self.fc1 = nn.Linear(64 * 14 * 14, 64)
        self.fc2 = nn.Linear(64, 1)

    def forward(self, x):
        x = self.pool1(torch.relu(self.conv1(x)))
```

```
        x = self.pool2(torch.relu(self.conv2(x)))
        x = x.view(-1, 64 * 14 * 14)
        x = self.dropout(x)
        x = torch.relu(self.fc1(x))
        return torch.sigmoid(self.fc2(x)).squeeze()

# Load model weights
model = EyeCNN()
model.load_state_dict(torch.load("eye_state_model_finetuned.pth",␣
 ↪map_location=torch.device('cpu')))
model.eval()

# === Dlib face/landmark detection ===
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
LEFT_EYE = list(range(36, 42))
RIGHT_EYE = list(range(42, 48))

# === Drowsiness parameters ===
CLOSED_THRESHOLD = 0.6
CLOSED_FRAMES_FOR_DROWSY = 7
CLOSED_FRAMES_FOR_BLINK = 2
closed_frame_count = 0
blink_count = 0

# === Preprocessing ===
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Grayscale(),
    transforms.Resize((64, 64)),
    transforms.ToTensor()
])

# === Extract eye from image ===
def extract_eye(img, eye_points, padding=35):
    points = np.array(eye_points, dtype=np.int32)
    x, y, w, h = cv2.boundingRect(points)
    x1 = max(x - padding, 0)
    y1 = max(y - padding, 0)
    x2 = min(x + w + padding, img.shape[1])
    y2 = min(y + h + padding, img.shape[0])
    return img[y1:y2, x1:x2]

# === Webcam feed ===
cap = cv2.VideoCapture(0)

while True:
```

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

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = detector(gray)

    for face in faces:
        landmarks = predictor(gray, face)
        left_eye_pts = [(landmarks.part(n).x, landmarks.part(n).y) for n in␣
↪LEFT_EYE]
        right_eye_pts = [(landmarks.part(n).x, landmarks.part(n).y) for n in␣
↪RIGHT_EYE]

        left_eye_img = extract_eye(frame, left_eye_pts)
        right_eye_img = extract_eye(frame, right_eye_pts)

        if left_eye_img.size > 0 and right_eye_img.size > 0:
            # Resize and preprocess
            left_tensor = transform(left_eye_img).unsqueeze(0)
            right_tensor = transform(right_eye_img).unsqueeze(0)

            # Model prediction
            with torch.no_grad():
                left_pred = model(left_tensor).item()
                right_pred = model(right_tensor).item()

            print(f"Left: {left_pred:.2f}, Right: {right_pred:.2f}")

            if left_pred < CLOSED_THRESHOLD and right_pred < CLOSED_THRESHOLD:
                closed_frame_count += 1
            else:
                if 1 <= closed_frame_count <= CLOSED_FRAMES_FOR_BLINK:
                    blink_count += 1
                closed_frame_count = 0

            # Display drowsy/awake status
            if closed_frame_count >= CLOSED_FRAMES_FOR_DROWSY:
                cv2.putText(frame, "DROWSY!", (50, 100), cv2.
↪FONT_HERSHEY_SIMPLEX, 2, (0,0,255), 3)
            else:
                cv2.putText(frame, "AWAKE!", (50, 100), cv2.
↪FONT_HERSHEY_SIMPLEX, 2, (0,255,0), 3)

            cv2.putText(frame, f"Blinks: {blink_count}", (50, 150), cv2.
↪FONT_HERSHEY_SIMPLEX, 1, (255,255,0), 2)
```

```
            # Preview eyes
            cv2.imshow("Left Eye", cv2.resize(left_eye_img, (128, 128)))
            cv2.imshow("Right Eye", cv2.resize(right_eye_img, (128, 128)))

    cv2.imshow("WakeMate - PyTorch Detection", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

```
Left: 0.76, Right: 0.62
Left: 0.85, Right: 0.71
Left: 0.84, Right: 0.66
Left: 0.83, Right: 0.78
Left: 0.86, Right: 0.82
Left: 0.83, Right: 0.71
Left: 0.94, Right: 0.72
Left: 0.88, Right: 0.76
Left: 0.92, Right: 0.84
Left: 0.95, Right: 0.82
Left: 0.97, Right: 0.97
Left: 0.97, Right: 0.97
Left: 0.96, Right: 0.93
Left: 0.96, Right: 0.90
Left: 0.96, Right: 0.89
Left: 0.98, Right: 0.92
Left: 0.96, Right: 0.92
Left: 0.98, Right: 0.95
Left: 0.97, Right: 0.95
Left: 0.97, Right: 0.92
Left: 0.96, Right: 0.91
Left: 0.25, Right: 0.42
Left: 0.21, Right: 0.44
Left: 0.45, Right: 0.44
Left: 0.92, Right: 0.82
Left: 0.95, Right: 0.90
Left: 0.93, Right: 0.89
Left: 0.96, Right: 0.93
Left: 0.26, Right: 0.34
Left: 0.78, Right: 0.58
Left: 0.90, Right: 0.79
Left: 0.93, Right: 0.88
Left: 0.93, Right: 0.84
Left: 0.90, Right: 0.81
Left: 0.23, Right: 0.44
Left: 0.44, Right: 0.46
```

Left: 0.95, Right: 0.83
Left: 0.96, Right: 0.88
Left: 0.97, Right: 0.90
Left: 0.97, Right: 0.89
Left: 0.21, Right: 0.38
Left: 0.76, Right: 0.61
Left: 0.95, Right: 0.87
Left: 0.96, Right: 0.90
Left: 0.94, Right: 0.85
Left: 0.23, Right: 0.39
Left: 0.25, Right: 0.39
Left: 0.19, Right: 0.31
Left: 0.30, Right: 0.23
Left: 0.96, Right: 0.74
Left: 0.97, Right: 0.83
Left: 0.97, Right: 0.91
Left: 0.97, Right: 0.89
Left: 0.89, Right: 0.79
Left: 0.24, Right: 0.41
Left: 0.21, Right: 0.32
Left: 0.19, Right: 0.40
Left: 0.08, Right: 0.10
Left: 0.19, Right: 0.37
Left: 0.18, Right: 0.36
Left: 0.05, Right: 0.24
Left: 0.20, Right: 0.34
Left: 0.48, Right: 0.42
Left: 0.93, Right: 0.59
Left: 0.96, Right: 0.67
Left: 0.95, Right: 0.77
Left: 0.95, Right: 0.79
Left: 0.86, Right: 0.79
Left: 0.84, Right: 0.85
Left: 0.78, Right: 0.79
Left: 0.88, Right: 0.75
Left: 0.86, Right: 0.82
Left: 0.94, Right: 0.72
Left: 0.91, Right: 0.71
Left: 0.31, Right: 0.22
Left: 0.23, Right: 0.35
Left: 0.24, Right: 0.38
Left: 0.24, Right: 0.38
Left: 0.20, Right: 0.41
Left: 0.18, Right: 0.35
Left: 0.22, Right: 0.40
Left: 0.15, Right: 0.25
Left: 0.94, Right: 0.59
Left: 0.98, Right: 0.72

Left: 0.97, Right: 0.71
Left: 0.96, Right: 0.75
Left: 0.76, Right: 0.77
Left: 0.33, Right: 0.16
Left: 0.26, Right: 0.36
Left: 0.28, Right: 0.38
Left: 0.23, Right: 0.41
Left: 0.20, Right: 0.30
Left: 0.25, Right: 0.32
Left: 0.57, Right: 0.48
Left: 0.97, Right: 0.68
Left: 0.97, Right: 0.80
Left: 0.98, Right: 0.81
Left: 0.96, Right: 0.86
Left: 0.88, Right: 0.70
Left: 0.90, Right: 0.80
Left: 0.87, Right: 0.81
Left: 0.94, Right: 0.87
Left: 0.91, Right: 0.87
Left: 0.96, Right: 0.78
Left: 0.94, Right: 0.83
Left: 0.97, Right: 0.89
Left: 0.96, Right: 0.84
Left: 0.20, Right: 0.34
Left: 0.19, Right: 0.39
Left: 0.19, Right: 0.36
Left: 0.22, Right: 0.35
Left: 0.25, Right: 0.35
Left: 0.19, Right: 0.38
Left: 0.18, Right: 0.42
Left: 0.18, Right: 0.30
Left: 0.80, Right: 0.57
Left: 0.94, Right: 0.78
Left: 0.94, Right: 0.82
Left: 0.94, Right: 0.78
Left: 0.58, Right: 0.65
Left: 0.59, Right: 0.59
Left: 0.89, Right: 0.55
Left: 0.92, Right: 0.55
Left: 0.96, Right: 0.64
Left: 0.22, Right: 0.32
Left: 0.22, Right: 0.24
Left: 0.23, Right: 0.34
Left: 0.28, Right: 0.33
Left: 0.20, Right: 0.44
Left: 0.22, Right: 0.36
Left: 0.29, Right: 0.21
Left: 0.39, Right: 0.17

```
Left: 0.85, Right: 0.50
Left: 0.88, Right: 0.39
Left: 0.70, Right: 0.26
Left: 0.83, Right: 0.32
Left: 0.97, Right: 0.67
Left: 0.97, Right: 0.77
Left: 0.82, Right: 0.42
Left: 0.91, Right: 0.64
Left: 0.79, Right: 0.68
Left: 0.89, Right: 0.75
Left: 0.79, Right: 0.45
```