In [1]:

```python
## Preparation

# Import the necessary libraries
import torch
import torch.nn as nn
from sklearn.model_selection import train_test_split
from torchvision import models, datasets, transforms
from torch.utils.data import DataLoader, Subset
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classificatio
import matplotlib.pyplot as plt
import numpy as np
import os
import optuna
import time

## Global Var

dataset_path = os.getcwd() + '/dataset/classified'

class_labels = ["Normal", "Cataract"]
```

In [2]:

```python
print(f"Is using CUDA? {torch.cuda.is_available()}")  # Should return True if CUDA
print(torch.version.cuda)  # Check the CUDA version PyTorch is using
print(torch.cuda.current_device()) # Check CUDA device used
```

```
Is using CUDA? True
12.6
0
```

In [3]:

```python
# Augmentation
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomRotation(15),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomCrop(224, padding=4),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize
])

# Load Dataset
ds = datasets.ImageFolder(root=dataset_path, transform=transform)

indices = list(range(len(ds)))
# labels = [ds.targets[i] for i in indices]

# Split into train and test dataset
train_indices, test_indices = train_test_split(indices, test_size=0.2, random_state

train_ds = Subset(ds, train_indices)
test_ds = Subset(ds, test_indices)

train_loader = DataLoader(train_ds, batch_size=64, shuffle=True, pin_memory=True, n
test_loader = DataLoader(test_ds, batch_size=64, shuffle=True, pin_memory=True, num
```

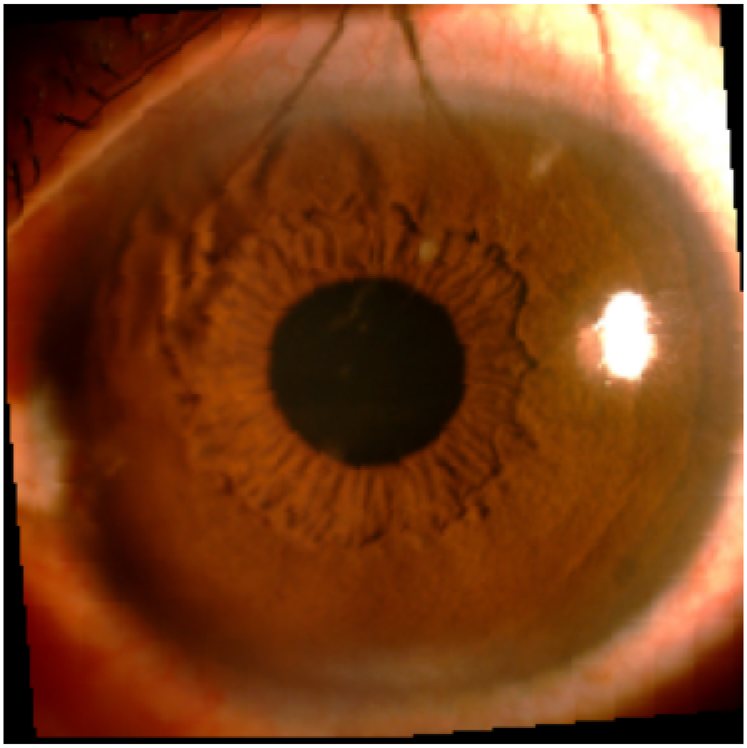
```python
total_samples  = len(train_ds) + len(test_ds)

print(f"Train size: {(len(train_ds) / total_samples) * 100:.2f}%, Test size: {(len(
print(f"Total samples: {total_samples}, Train size: {len(train_ds)}, Test size: {le
```

```
Train size: 79.98%, Test size: 20.02%
Total samples: 1159, Train size: 927, Test size: 232
```

In [4]:

```python
def denormalize(tensor, mean=None, std=None):
    if std is None:
        std = [0.5, 0.5, 0.5]
    if mean is None:
        mean = [0.5, 0.5, 0.5]
    mean = torch.tensor(mean).view(3, 1, 1)
    std = torch.tensor(std).view(3, 1, 1)
    return tensor * std + mean  # Reverse normalization

# Get a batch of images
dataiter = iter(train_loader)
images, labels = next(dataiter)

# Select one image
img = images[0]
label = labels[0].item()

# Denormalize image
img = denormalize(img)

# Convert from Tensor (C, H, W) to NumPy (H, W, C)
img = np.transpose(img.numpy(), (1, 2, 0))

# Plot the image
plt.imshow(img)
plt.title(f"Label: {class_labels[label]}")  # Display label
plt.axis("off")
plt.show()
```

Label: Normal



In [5]:
```
# Optuna Hyperparameter

best_trial = None
best_model = None

torch.backends.cudnn.benchmark = True

device_name = "cuda" if torch.cuda.is_available() else "cpu"
use_amp = device_name == "cuda"


def objective(trial: optuna.Trial) -> float:
    device = torch.device(device_name)

    # Define hyperparameters
    lr = trial.suggest_float("lr", 1e-3, 1e-1, log=True)
    dropout_rate = trial.suggest_float("dropout", 0.2, 0.5)
    optimizer_name = trial.suggest_categorical("optimizer", ["AdamW", "SGD"])
    num_epochs = trial.suggest_int("num_epochs", 5, 10)  # Fixed at 5 epochs

    model = models.efficientnet_b0(progress=True, weights=models.EfficientNet_B0_We

    for param in model.features[:-2].parameters():
        param.requires_grad = False

    number_of_features = model.classifier[1].in_features
    model.classifier = nn.Sequential(
        nn.Linear(number_of_features, 128),
        nn.ReLU(),
        nn.Dropout(dropout_rate),
        nn.Linear(128, 1)
```

```
    ).to(device)

    model.to(device)

    # Define loss and optimizer
    criterion = nn.BCEWithLogitsLoss()

    optimizer = torch.optim.AdamW(model.parameters(), lr=lr, weight_decay=1e-4)
    if optimizer_name == "SGD":
        optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.9)

    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=10)

    train_losses = []
    train_accuracies = []

    scaler = torch.amp.GradScaler(device=device_name)

    # Training Loop
    for epoch in range(num_epochs):

        model.train()
        total_loss, correct, total = 0, 0, 0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.float().to(device).unsqueeze

            optimizer.zero_grad()

            with torch.amp.autocast("cuda", enabled=use_amp):
                outputs = model(images)
                loss = criterion(outputs, labels)

            scaler.scale(loss).backward()
            scaler.step(optimizer)
            scaler.update()

            total_loss += loss.item()
            preds = (torch.sigmoid(outputs) > 0.5).float()
            correct += (preds == labels).sum().item()
            total += labels.size(0)

        scheduler.step()

        epoch_loss = total_loss / len(train_loader)
        epoch_acc = correct / total * 100

        train_losses.append(epoch_loss)
        train_accuracies.append(epoch_acc)

        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}, Accuracy: {

        trial.set_user_attr("train_losses", train_losses)
        trial.set_user_attr("train_accuracies", train_accuracies)

        # Pruning: Stop bad trials early
```

```python
        trial.report(epoch_acc, epoch)
        if trial.should_prune():
            raise optuna.exceptions.TrialPruned()

    # Evaluate Model
    model.eval()
    correct, total = 0, 0
    trial_preds = []
    trial_labels = []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.float().to(device).unsqueeze
            outputs = model(images)
            preds = (torch.sigmoid(outputs) > 0.5).float()

            correct += (preds == labels).sum().item()
            total += labels.size(0)

            trial_preds.extend(preds.cpu().numpy())
            trial_labels.extend(labels.cpu().numpy())

    test_acc = correct / total * 100
    print(f"🎯 Test Accuracy: {test_acc:.2f}%")

    # **Save best model globally**
    global best_model, best_trial

    try:
        best_trial = trial.study.best_trial
        best_value = trial.study.best_value
    except ValueError:
        best_value = float('-inf')

    if best_model is None or (best_trial is not None and test_acc > best_value):
        best_model = model.state_dict()

    best_acc = trial.study.user_attrs.get("best_accuracy", 0)

    if test_acc > best_acc:
        study.set_user_attr("best_accuracy", test_acc)

        trial_preds = np.array(trial_preds).flatten().tolist()
        trial_labels = np.array(trial_labels).flatten().tolist()

        study.set_user_attr("best_preds", trial_preds)
        study.set_user_attr("best_labels", trial_labels)

    return test_acc
```

In [6]:
```python
study = optuna.create_study(
    direction="maximize",
    study_name=f"hyperparam cataract classifier_{int(time.time())}",
    pruner=optuna.pruners.MedianPruner(),
    storage="sqlite:///optuna.db",
    load_if_exists=True
```

```
)

study.optimize(
    objective,
    n_trials=20,
    n_jobs=4,
    show_progress_bar=True
)

acc = study.best_value

print("\nBest Hyperparameters:", study.best_params)
print("\nAccuracy:", acc)
```

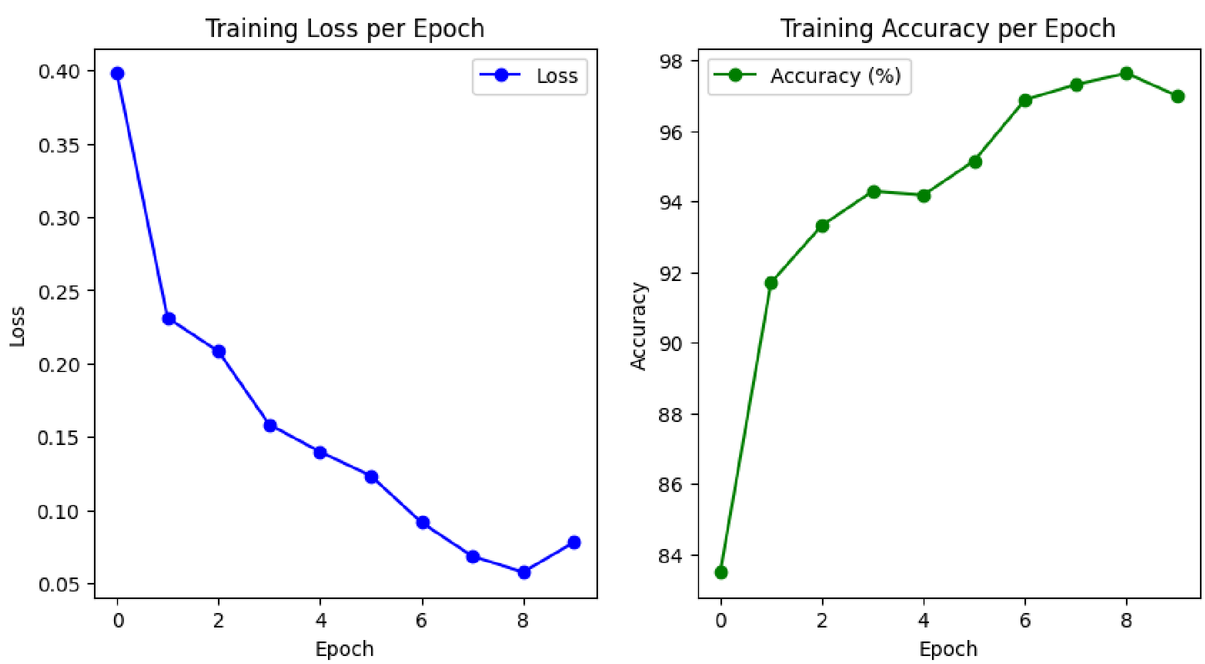
```
[I 2025-03-13 02:15:18,379] A new study created in RDB with name: hyperparam catarac
t classifier_1741806917
  0%|          | 0/20 [00:00<?, ?it/s]
```

```
Epoch [1/10], Loss: 0.3981, Accuracy: 83.50%
Epoch [1/6], Loss: 1.1209, Accuracy: 70.77%
Epoch [1/10], Loss: 0.6950, Accuracy: 49.08%
Epoch [1/10], Loss: 0.3609, Accuracy: 81.77%
Epoch [2/10], Loss: 0.2310, Accuracy: 91.69%
Epoch [2/6], Loss: 0.3079, Accuracy: 87.70%
Epoch [2/10], Loss: 0.6490, Accuracy: 70.33%
Epoch [2/10], Loss: 0.2020, Accuracy: 91.91%
Epoch [3/6], Loss: 0.2354, Accuracy: 90.83%
Epoch [3/10], Loss: 0.2082, Accuracy: 93.31%
Epoch [3/10], Loss: 0.6113, Accuracy: 70.66%
Epoch [3/10], Loss: 0.1619, Accuracy: 94.93%
Epoch [4/6], Loss: 0.1987, Accuracy: 92.77%
Epoch [4/10], Loss: 0.1583, Accuracy: 94.28%
Epoch [4/10], Loss: 0.5920, Accuracy: 70.66%
Epoch [4/10], Loss: 0.1492, Accuracy: 94.17%
Epoch [5/10], Loss: 0.1396, Accuracy: 94.17%
Epoch [5/10], Loss: 0.5830, Accuracy: 70.66%
Epoch [5/6], Loss: 0.2031, Accuracy: 94.17%
Epoch [5/10], Loss: 0.1069, Accuracy: 96.33%
Epoch [6/10], Loss: 0.1233, Accuracy: 95.15%
Epoch [6/10], Loss: 0.5658, Accuracy: 70.66%
Epoch [6/6], Loss: 0.1655, Accuracy: 93.42%
Epoch [6/10], Loss: 0.0855, Accuracy: 97.09%
🎯 Test Accuracy: 93.10%
[I 2025-03-13 02:19:41,030] Trial 0 finished with value: 93.10344827586206 and param
eters: {'lr': 0.02989743682562383, 'dropout': 0.41864583212524387, 'optimizer': 'Ada
mW', 'num_epochs': 6}. Best is trial 0 with value: 93.10344827586206.
Epoch [7/10], Loss: 0.0914, Accuracy: 96.87%
Epoch [7/10], Loss: 0.5619, Accuracy: 70.66%
Epoch [7/10], Loss: 0.0821, Accuracy: 96.87%
Epoch [1/8], Loss: 0.3949, Accuracy: 81.12%
Epoch [8/10], Loss: 0.0684, Accuracy: 97.30%
Epoch [8/10], Loss: 0.5485, Accuracy: 70.66%
Epoch [8/10], Loss: 0.0629, Accuracy: 97.73%
Epoch [2/8], Loss: 0.2255, Accuracy: 90.18%
Epoch [9/10], Loss: 0.5536, Accuracy: 70.66%
Epoch [9/10], Loss: 0.0575, Accuracy: 97.63%
Epoch [9/10], Loss: 0.0544, Accuracy: 98.17%
Epoch [10/10], Loss: 0.5460, Accuracy: 70.66%
Epoch [3/8], Loss: 0.1722, Accuracy: 93.64%
Epoch [10/10], Loss: 0.0779, Accuracy: 96.98%
Epoch [10/10], Loss: 0.0515, Accuracy: 98.60%
🎯 Test Accuracy: 75.00%
[I 2025-03-13 02:21:46,974] Trial 3 finished with value: 75.0 and parameters: {'lr':
0.0020507919934963925, 'dropout': 0.41314315121606576, 'optimizer': 'SGD', 'num_epoc
hs': 10}. Best is trial 0 with value: 93.10344827586206.
🎯 Test Accuracy: 94.40%
[I 2025-03-13 02:21:47,898] Trial 1 finished with value: 94.39655172413794 and param
eters: {'lr': 0.008650262291857513, 'dropout': 0.3336420482173461, 'optimizer': 'Ada
mW', 'num_epochs': 10}. Best is trial 1 with value: 94.39655172413794.
Epoch [4/8], Loss: 0.1351, Accuracy: 94.71%
🎯 Test Accuracy: 92.24%
[I 2025-03-13 02:21:58,049] Trial 2 finished with value: 92.24137931034483 and param
eters: {'lr': 0.003966599858683413, 'dropout': 0.39308162601981733, 'optimizer': 'Ad
amW', 'num_epochs': 10}. Best is trial 1 with value: 94.39655172413794.
```

```
Epoch [1/7], Loss: 0.5831, Accuracy: 68.93%
Epoch [1/6], Loss: 0.6680, Accuracy: 63.86%
Epoch [5/8], Loss: 0.1137, Accuracy: 95.36%
Epoch [1/5], Loss: 3.9171, Accuracy: 55.34%
Epoch [2/7], Loss: 0.3753, Accuracy: 82.74%
Epoch [2/6], Loss: 0.5928, Accuracy: 70.66%
Epoch [6/8], Loss: 0.0817, Accuracy: 97.52%
Epoch [2/5], Loss: 0.5025, Accuracy: 76.27%
Epoch [3/7], Loss: 0.2303, Accuracy: 91.26%
Epoch [3/6], Loss: 0.5432, Accuracy: 70.66%
Epoch [7/8], Loss: 0.0686, Accuracy: 97.52%
Epoch [3/5], Loss: 0.2874, Accuracy: 89.32%
Epoch [4/7], Loss: 0.1878, Accuracy: 92.66%
Epoch [4/6], Loss: 0.5173, Accuracy: 70.66%
Epoch [8/8], Loss: 0.0629, Accuracy: 97.30%
Epoch [4/5], Loss: 0.2010, Accuracy: 92.13%
🎯 Test Accuracy: 92.67%
[I 2025-03-13 02:24:24,624] Trial 4 finished with value: 92.67241379310344 and param
eters: {'lr': 0.0037016668606250436, 'dropout': 0.4264276214041657, 'optimizer': 'Ad
amW', 'num_epochs': 8}. Best is trial 1 with value: 94.39655172413794.
Epoch [5/7], Loss: 0.1427, Accuracy: 95.04%
Epoch [5/6], Loss: 0.4955, Accuracy: 71.31%
[I 2025-03-13 02:24:27,893] Trial 6 pruned.
Epoch [5/5], Loss: 0.2025, Accuracy: 92.45%
[I 2025-03-13 02:24:48,061] Trial 7 pruned.
Epoch [1/9], Loss: 3.5660, Accuracy: 61.92%
[I 2025-03-13 02:24:58,281] Trial 8 pruned.
Epoch [6/7], Loss: 0.1168, Accuracy: 95.25%
Epoch [1/10], Loss: 0.5652, Accuracy: 70.44%
[I 2025-03-13 02:25:00,242] Trial 9 pruned.
Epoch [1/6], Loss: 0.6901, Accuracy: 53.51%
[I 2025-03-13 02:25:23,363] Trial 10 pruned.
Epoch [1/7], Loss: 0.4230, Accuracy: 78.53%
[I 2025-03-13 02:25:30,475] Trial 11 pruned.
Epoch [7/7], Loss: 0.1106, Accuracy: 95.47%
[I 2025-03-13 02:25:31,643] Trial 5 pruned.
Epoch [1/9], Loss: 0.6835, Accuracy: 55.77%
[I 2025-03-13 02:25:33,617] Trial 12 pruned.
Epoch [1/8], Loss: 0.4550, Accuracy: 79.61%
[I 2025-03-13 02:25:58,307] Trial 13 pruned.
Epoch [1/8], Loss: 0.7141, Accuracy: 72.60%
[I 2025-03-13 02:26:02,427] Trial 14 pruned.
Epoch [1/8], Loss: 0.5978, Accuracy: 78.10%
[I 2025-03-13 02:26:03,555] Trial 15 pruned.
Epoch [1/5], Loss: 0.6299, Accuracy: 71.84%
[I 2025-03-13 02:26:07,526] Trial 16 pruned.
Epoch [1/5], Loss: 0.6042, Accuracy: 73.46%
[I 2025-03-13 02:26:30,627] Trial 17 pruned.
Epoch [1/5], Loss: 0.9715, Accuracy: 72.92%
[I 2025-03-13 02:26:33,234] Trial 18 pruned.
Epoch [1/5], Loss: 1.3657, Accuracy: 64.29%
[I 2025-03-13 02:26:33,883] Trial 19 pruned.

Best Hyperparameters: {'lr': 0.008650262291857513, 'dropout': 0.3336420482173461, 'o
ptimizer': 'AdamW', 'num_epochs': 10}
```

```
Accuracy: 94.39655172413794
```

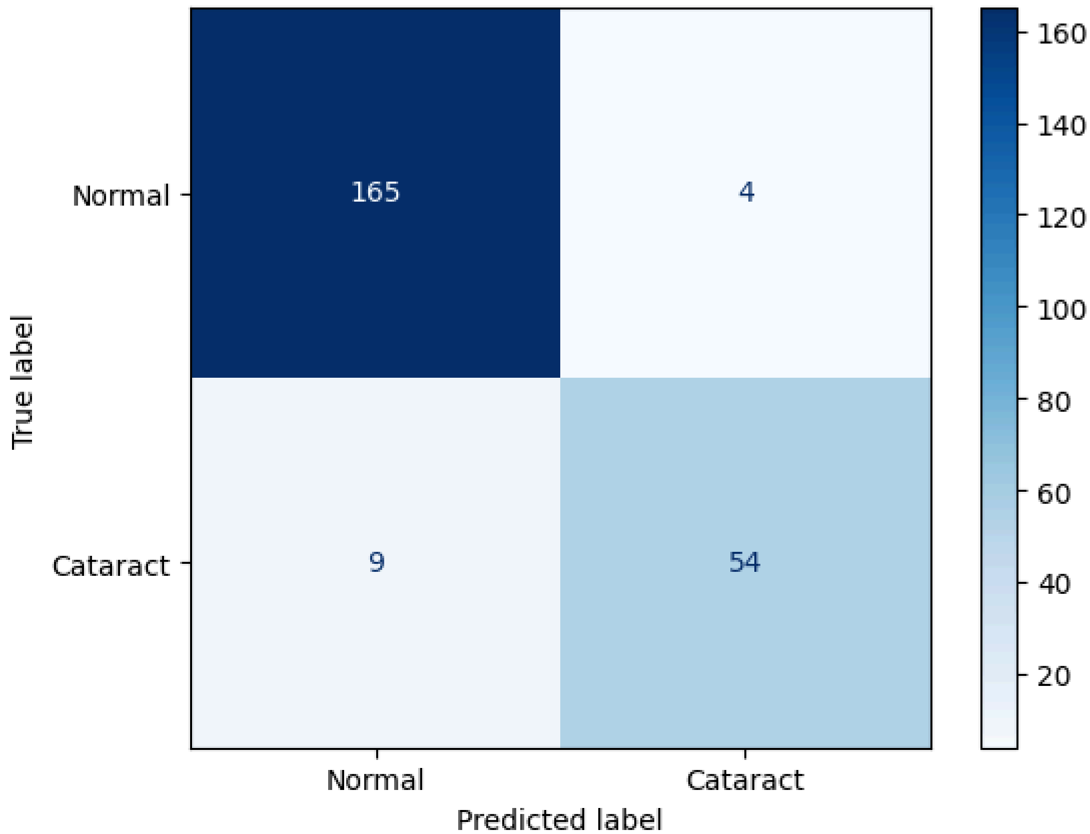
In [7]:
```python
best_trial = study.best_trial
best_train_losses = best_trial.user_attrs.get("train_losses", [])
best_train_accuracies = best_trial.user_attrs.get("train_accuracies", [])

plt.figure(figsize=(10, 5))

# Loss Graph
plt.subplot(1, 2, 1)
plt.plot(best_train_losses, label="Loss", marker="o", linestyle="-", color="b")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training Loss per Epoch")
plt.legend()

# Accuracy Graph
plt.subplot(1, 2, 2)
plt.plot(best_train_accuracies, label="Accuracy (%)", marker="o", linestyle="-", co
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Training Accuracy per Epoch")
plt.legend()

plt.show()
```



In [8]:
```python
best_preds = np.array(study.user_attrs.get("best_preds", []))
best_labels = np.array(study.user_attrs.get("best_labels", []))

conf_matrix = confusion_matrix(best_preds, best_labels)
ConfusionMatrixDisplay(conf_matrix, display_labels=class_labels).plot(cmap=plt.cm.B
cr = classification_report(best_preds, best_labels)
```

```python
print(f"Accuracy: {acc:.2f}%")
print(cr)
```

```
Accuracy: 94.40%
              precision    recall  f1-score   support

         0.0       0.95      0.98      0.96       169
         1.0       0.93      0.86      0.89        63

    accuracy                           0.94       232
   macro avg       0.94      0.92      0.93       232
weighted avg       0.94      0.94      0.94       232
```



In [9]:

```python
# # save
output_model_path = f"output/checkpoint-{acc}-hyperparam.pth"

torch.save({
    "model_state_dict": study.user_attrs.get("best_model_state"),  # Best model wei
    "optimizer_state_dict": study.user_attrs.get("best_optimizer_state"),  # Best o
    "best_hyperparameters": study.best_params,
    "best_accuracy": study.best_value,
    "best_train_losses": study.user_attrs.get("best_train_losses", []),  # Best tra
    "best_train_accuracies": study.user_attrs.get("best_train_accuracies", []),  #
}, output_model_path)
```

In [10]:

```python
# checkpoint = torch.load("checkpoint.pth")
# model.load_state_dict(checkpoint['model_state_dict'])
# optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
# model.eval()
```