

fbrwq7gm3

November 13, 2025

```
[18]: import os
```

```
[19]: from IPython.display import Image, display
```

```
[20]: import numpy as np
```

```
[21]: pip install torch torchvision torchaudio
```

```
Requirement already satisfied: torch in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (2.9.1)  
Requirement already satisfied: torchvision in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (0.24.1)  
Requirement already satisfied: torchaudio in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (2.9.1)  
Requirement already satisfied: filelock in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (from torch) (3.13.1)  
Requirement already satisfied: typing-extensions>=4.10.0 in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (from torch) (4.11.0)  
Requirement already satisfied: sympy>=1.13.3 in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (from torch) (1.14.0)  
Requirement already satisfied: networkx>=2.5.1 in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (from torch) (3.3)  
Requirement already satisfied: jinja2 in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (from torch) (3.1.4)  
Requirement already satisfied: fsspec>=0.8.5 in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (from torch) (2024.6.1)  
Requirement already satisfied: setuptools in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (from torch) (75.1.0)  
Requirement already satisfied: numpy in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (from torchvision) (1.26.4)  
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (from torchvision) (10.4.0)  
Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (from sympy>=1.13.3->torch) (1.3.0)  
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\mohammed al  
malki\anaconda3\lib\site-packages (from jinja2->torch) (2.1.3)  
Note: you may need to restart the kernel to use updated packages.
```

```
[23]: import torch
from torchvision import datasets, transforms as T, models
from torch import nn, optim
from torch.utils.data import DataLoader
```

```
[24]: import warnings
warnings.filterwarnings("ignore")
from tqdm import tqdm
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, r
↪ recall_score, f1_score, classification_report
```

```
[25]: import matplotlib.pyplot as plt
```

```
[26]: import numpy as np
```

```
[27]: import seaborn as sns
from sklearn.metrics import roc_curve
```

```
[28]: import matplotlib.pyplot as plt
from sklearn.metrics import auc
```

```
[75]: from PIL import Image
```

```
[49]: import os
```

```
[51]: dataset_path = r'C:\Users\Mohammed Al Malki\Downloads\dataset'
```

```
[52]: if os.path.exists(dataset_path):
    print(f"                : {dataset_path}")
```

```
                : C:\Users\Mohammed Al Malki\Downloads\dataset
```

```
[ ]:
```

```
[127]: path = r'C:\Users\Mohammed Al Malki\Downloads\dataset\newdataset'
if os.path.exists(path):
    for item in os.listdir(path):
        full_path = os.path.join(path, item)
        if os.path.isdir(full_path):
            count = len([f for f in os.listdir(full_path) if f.lower().
↪endswith(('.png', '.jpg', '.jpeg'))])
            print(f" {item} ({count} images)")
        else:
            print(f" {item}")
```

```
cataract (1038 images)
diabetic_retinopathy (1098 images)
```

```
glaucoma (1007 images)
normal (1074 images)
```

```
[103]: for i, folder_path in enumerate(folder_paths):
        if not os.path.isdir(folder_path):
            print(f" path is not found: {folder_path}")
            continue
```

```
[104]: for class_name in os.listdir(input_root):
        class_path = os.path.join(input_root, class_name)

        if not os.path.isdir(class_path) or class_name == 'masks':
            continue
```

```
[107]: input_root = 'dataset/'
        output_root = 'output/'
        resize_size = (256, 256) # Change size as needed

        # Create output root if it doesn't exist
        os.makedirs(output_root, exist_ok=True)

        # Iterate through each class folder
        for class_name in os.listdir(input_root):
            class_path = os.path.join(input_root, class_name)

            # Skip if it's not a directory or if it's the 'masks' folder
            if not os.path.isdir(class_path) or class_name == 'masks':
                continue

            # Create output path for each class
            output_class_path = os.path.join(output_root, class_name)
            os.makedirs(output_class_path, exist_ok=True)

            # Process each image inside the class folder
            for image_name in os.listdir(class_path):
                if image_name.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.
                ↪tiff')):

                    input_image_path = os.path.join(class_path, image_name)
                    output_image_name = os.path.splitext(image_name)[0] + '.jpg'
                    output_image_path = os.path.join(output_class_path,
                    ↪output_image_name)

                    # Try processing the image
                    try:
                        with Image.open(input_image_path) as img:
```

```

        img = img.convert('RGB')
        img = img.resize(resize_size)
        img.save(output_image_path, 'JPEG')
        print(f"Processed: {input_image_path} ->↳
↳{output_image_path}")

    except Exception as e:
        print(f"Failed to process {input_image_path}: {e}")

```

```

[116]: base_dir = 'newdataset'
os.makedirs(base_dir, exist_ok=True)

classes = ['normal', 'glaucoma', 'diabetic_retinopathy', 'cataract']

for cls in classes:
    class_dir = os.path.join(base_dir, cls)
    os.makedirs(class_dir, exist_ok=True)

print("Folders created successfully inside newdataset!")

```

Folders created successfully inside newdataset!

```

[108]: import os
import shutil
from sklearn.model_selection import train_test_split

```

```

[118]: target_root = 'newdataset'
classes = ['normal', 'glaucoma', 'diabetic_retinopathy', 'cataract']

os.makedirs(target_root, exist_ok=True)

for cls in classes:
    os.makedirs(os.path.join(target_root, cls), exist_ok=True)

```

```

[119]: print("Contents of 'newdataset':")
print(os.listdir("newdataset"))

```

Contents of 'newdataset':

['cataract', 'diabetic_retinopathy', 'glaucoma', 'normal']

```

[128]: path = r'C:\Users\Mohammed Al Malki\Downloads\dataset\newdataset'
if os.path.exists(path):
    for item in os.listdir(path):
        full_path = os.path.join(path, item)
        if os.path.isdir(full_path):
            count = len([f for f in os.listdir(full_path) if f.lower().
↳endswith(('.png', '.jpg', '.jpeg'))])

```

```

        print(f" {item} ({count} images)")
    else:
        print(f" {item}")

```

```

cataract (1038 images)
diabetic_retinopathy (1098 images)
glaucoma (1007 images)
normal (1074 images)

```

```

[ ]: import os
import torch
from torch import nn, optim
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms, models

data_dir = "newdataset"
batch_size = 32
num_epochs = 10
learning_rate = 1e-4
val_ratio = 0.2

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

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

full_dataset = datasets.ImageFolder(data_dir, transform=transform)
num_classes = len(full_dataset.classes)

val_size = int(len(full_dataset) * val_ratio)
train_size = len(full_dataset) - val_size

train_dataset, val_dataset = random_split(full_dataset, [train_size, val_size])

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True,
    ↪num_workers=2)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False,
    ↪num_workers=2)

model = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
model.fc = nn.Linear(model.fc.in_features, num_classes)

```

```

model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0

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

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images.size(0)

    epoch_loss = running_loss / len(train_loader.dataset)

    model.eval()
    correct = 0
    total = 0

    with torch.no_grad():
        for images, labels in val_loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            _, preds = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (preds == labels).sum().item()

    val_acc = 100.0 * correct / total
    print(f"Epoch {epoch + 1}/{num_epochs}, Loss: {epoch_loss:.4f}, Val Acc: ␣
↪{val_acc:.2f}%")

class_names = full_dataset.classes
print("Classes:", class_names)

```

Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to
 C:\Users\Mohammed Al Malki\.cache\torch\hub\checkpoints\resnet18-f37072fd.pth
 100% | 44.7M/44.7M [00:05<00:00, 9.23MB/s]

Epoch 1/10, Loss: 0.3677, Val Acc: 88.73%
Epoch 2/10, Loss: 0.1533, Val Acc: 89.92%
Epoch 3/10, Loss: 0.0710, Val Acc: 90.04%

```
[ ]: import os
import torch
from torch import nn, optim
from torch.utils.data import DataLoader, Subset
from torchvision import datasets, transforms, models

data_dir = "newdataset"
batch_size = 32
num_epochs = 10
learning_rate = 1e-4
val_ratio = 0.2

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

train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.
↪02),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

val_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

base_dataset = datasets.ImageFolder(data_dir)
num_classes = len(base_dataset.classes)

num_samples = len(base_dataset)
val_size = int(num_samples * val_ratio)
train_size = num_samples - val_size
```

```

indices = torch.randperm(num_samples)
train_indices = indices[:train_size]
val_indices = indices[train_size:]

train_dataset_full = datasets.ImageFolder(data_dir, transform=train_transform)
val_dataset_full = datasets.ImageFolder(data_dir, transform=val_transform)

train_dataset = Subset(train_dataset_full, train_indices)
val_dataset = Subset(val_dataset_full, val_indices)

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True,
    ↪num_workers=2)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False,
    ↪num_workers=2)

model = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
model.fc = nn.Linear(model.fc.in_features, num_classes)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0

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

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images.size(0)

    epoch_loss = running_loss / len(train_loader.dataset)

    model.eval()
    correct = 0
    total = 0

    with torch.no_grad():
        for images, labels in val_loader:
            images = images.to(device)

```

```

        labels = labels.to(device)

        outputs = model(images)
        _, preds = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (preds == labels).sum().item()

    val_acc = 100.0 * correct / total
    print(f"Epoch {epoch + 1}/{num_epochs}, Loss: {epoch_loss:.4f}, Val Acc:␣
↪{val_acc:.2f}%")

class_names = base_dataset.classes
print("Classes:", class_names)

```

```

[ ]: from torchvision.models import efficientnet_b0, EfficientNet_B0_Weights

weights = EfficientNet_B0_Weights.DEFAULT
model = efficientnet_b0(weights=weights)

for param in model.features.parameters():
    param.requires_grad = False

in_features = model.classifier[1].in_features
model.classifier[1] = nn.Linear(in_features, num_classes)

model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.classifier.parameters(), lr=learning_rate)

```

```

[140]: # Lists to store metrics for plotting
train_losses = []
val_accuracies = []

for epoch in range(num_epochs):
    # Training phase
    model.train()
    running_loss = 0.0

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

        # Zero the gradients
        optimizer.zero_grad()

        # Forward pass

```

```

        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward pass and optimize
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images.size(0)

    # Calculate average training loss
    epoch_loss = running_loss / len(train_loader.dataset)
    train_losses.append(epoch_loss)

    # Validation phase
    model.eval()
    correct = 0
    total = 0

    with torch.no_grad():
        for images, labels in val_loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            _, preds = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (preds == labels).sum().item()

    val_acc = 100.0 * correct / total
    val_accurrencies.append(val_acc)

    print(f"Epoch {epoch + 1}/{num_epochs}, Loss: {epoch_loss:.4f}, Val Acc: {val_acc:.2f}%")

# Print class names after training
class_names = train_dataset.classes
print("\nTraining completed!")
print("Classes:", class_names)
print(f"Number of classes: {len(class_names)}")

```

KeyboardInterrupt

Traceback (most recent call last)

Cell In[140], line 18

```
15 optimizer.zero_grad()
```

```
17 # Forward pass
```

```
----> 18 outputs = model(images)
```

```
19 loss = criterion(outputs, labels)
```

```

    21 # Backward pass and optimize

File ~\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1775, in Module.
    ↪ _wrapped_call_impl(self, *args, **kwargs)
    1773     return self._compiled_call_impl(*args, **kwargs) # type:␣
    ↪ ignore[misc]
    1774 else:
-> 1775     return self._call_impl(*args, **kwargs)

File ~\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1786, in Module.
    ↪ _call_impl(self, *args, **kwargs)
    1781 # If we don't have any hooks, we want to skip the rest of the logic in
    1782 # this function, and just call forward.
    1783 if not (self._backward_hooks or self._backward_pre_hooks or self.
    ↪ _forward_hooks or self._forward_pre_hooks
    1784         or _global_backward_pre_hooks or _global_backward_hooks
    1785         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1786     return forward_call(*args, **kwargs)
    1788 result = None
    1789 called_always_called_hooks = set()

File ~\anaconda3\Lib\site-packages\torchvision\models\resnet.py:285, in ResNet.
    ↪ forward(self, x)
    284 def forward(self, x: Tensor) -> Tensor:
--> 285     return self._forward_impl(x)

File ~\anaconda3\Lib\site-packages\torchvision\models\resnet.py:273, in ResNet.
    ↪ _forward_impl(self, x)
    270 x = self.relu(x)
    271 x = self.maxpool(x)
--> 273 x = self.layer1(x)
    274 x = self.layer2(x)
    275 x = self.layer3(x)

File ~\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1775, in Module.
    ↪ _wrapped_call_impl(self, *args, **kwargs)
    1773     return self._compiled_call_impl(*args, **kwargs) # type:␣
    ↪ ignore[misc]
    1774 else:
-> 1775     return self._call_impl(*args, **kwargs)

File ~\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1786, in Module.
    ↪ _call_impl(self, *args, **kwargs)
    1781 # If we don't have any hooks, we want to skip the rest of the logic in
    1782 # this function, and just call forward.
    1783 if not (self._backward_hooks or self._backward_pre_hooks or self.
    ↪ _forward_hooks or self._forward_pre_hooks
    1784         or _global_backward_pre_hooks or _global_backward_hooks

```

```

1785         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1786     return forward_call(*args, **kwargs)
1788 result = None
1789 called_always_called_hooks = set()

File ~\anaconda3\Lib\site-packages\torch\nn\modules\container.py:250, in
↳ Sequential.forward(self, input)
    246 """
    247 Runs the forward pass.
    248 """
    249 for module in self:
--> 250     input = module(input)
    251 return input

File ~\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1775, in Module.
↳ _wrapped_call_impl(self, *args, **kwargs)
    1773     return self._compiled_call_impl(*args, **kwargs) # type:
↳ ignore[misc]
    1774 else:
-> 1775     return self._call_impl(*args, **kwargs)

File ~\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1786, in Module.
↳ _call_impl(self, *args, **kwargs)
    1781 # If we don't have any hooks, we want to skip the rest of the logic in
    1782 # this function, and just call forward.
    1783 if not (self._backward_hooks or self._backward_pre_hooks or self.
↳ _forward_hooks or self._forward_pre_hooks
    1784         or _global_backward_pre_hooks or _global_backward_hooks
    1785         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1786     return forward_call(*args, **kwargs)
    1788 result = None
    1789 called_always_called_hooks = set()

File ~\anaconda3\Lib\site-packages\torchvision\models\resnet.py:96, in
↳ BasicBlock.forward(self, x)
    93 out = self.bn1(out)
    94 out = self.relu(out)
---> 96 out = self.conv2(out)
    97 out = self.bn2(out)
    99 if self.downsample is not None:

File ~\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1775, in Module.
↳ _wrapped_call_impl(self, *args, **kwargs)
    1773     return self._compiled_call_impl(*args, **kwargs) # type:
↳ ignore[misc]
    1774 else:
-> 1775     return self._call_impl(*args, **kwargs)

```

File ~\anaconda3\Lib\site-packages\torch\nn\modules\module.py:1786, in Module.

```
→ _call_impl(self, *args, **kwargs)
    1781 # If we don't have any hooks, we want to skip the rest of the logic in
    1782 # this function, and just call forward.
    1783 if not (self._backward_hooks or self._backward_pre_hooks or self.
→ _forward_hooks or self._forward_pre_hooks
    1784         or _global_backward_pre_hooks or _global_backward_hooks
    1785         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1786     return forward_call(*args, **kwargs)
    1788 result = None
    1789 called_always_called_hooks = set()
```

File ~\anaconda3\Lib\site-packages\torch\nn\modules\conv.py:548, in Conv2d.

```
→ forward(self, input)
    547 def forward(self, input: Tensor) -> Tensor:
-> 548     return self._conv_forward(input, self.weight, self.bias)
```

File ~\anaconda3\Lib\site-packages\torch\nn\modules\conv.py:543, in Conv2d.

```
→ _conv_forward(self, input, weight, bias)
    531 if self.padding_mode != "zeros":
    532     return F.conv2d(
    533         F.pad(
    534             input, self._reversed_padding_repeated_twice, mode=self.
→ padding_mode
    (...
    541         self.groups,
    542     )
-> 543 return F.conv2d(
    544     input, weight, bias, self.stride, self.padding, self.dilation, self
→ groups
    545 )
```

KeyboardInterrupt:

```
[ ]: torch.save(model.state_dict(), "efficientnet_b0_model.pth")
print("Model saved as efficientnet_b0_model.pth")
```

[141]: *# Initialize lists to track metrics*

```
train_losses = []
val_accuracies = []

for epoch in range(num_epochs):
    # Training phase
    model.train()
    running_loss = 0.0
    train_correct = 0
    train_total = 0
```

```

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

    optimizer.zero_grad()
    outputs = model(images)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()

    running_loss += loss.item() * images.size(0)

    # Calculate training accuracy
    _, predicted = torch.max(outputs.data, 1)
    train_total += labels.size(0)
    train_correct += (predicted == labels).sum().item()

epoch_loss = running_loss / len(train_loader.dataset)
train_accuracy = 100.0 * train_correct / train_total
train_losses.append(epoch_loss)

# Validation phase
model.eval()
val_correct = 0
val_total = 0

with torch.no_grad():
    for images, labels in val_loader:
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        val_total += labels.size(0)
        val_correct += (predicted == labels).sum().item()

val_accuracy = 100.0 * val_correct / val_total
val_accuracies.append(val_accuracy)

print(f"Epoch [{epoch+1}/{num_epochs}]")
print(f"  Training Loss: {epoch_loss:.4f}, Training Accuracy: ⬇
↪{train_accuracy:.2f}%")
print(f"  Validation Accuracy: {val_accuracy:.2f}%")
print("-" * 50)

# Get class names

```

```

try:
    class_names = base_dataset.classes
except:
    try:
        class_names = train_dataset.classes
    except:
        class_names = [f"Class_{i}" for i in range(num_classes)]

print("Classes:", class_names)

```

```

-----
KeyboardInterrupt                                Traceback (most recent call last)
Cell In[141], line 19
     17 outputs = model(images)
     18 loss = criterion(outputs, labels)
--> 19 loss.backward()
     20 optimizer.step()
     22 running_loss += loss.item() * images.size(0)

File ~\anaconda3\Lib\site-packages\torch\tensor.py:625, in Tensor.
backward(self, gradient, retain_graph, create_graph, inputs)
     615 if has_torch_function_unary(self):
     616     return handle_torch_function(
     617         Tensor.backward,
     618         (self,),
     (...
     623         inputs=inputs,
     624     )
--> 625 torch.autograd.backward(
     626     self, gradient, retain_graph, create_graph, inputs=inputs
     627 )

File ~\anaconda3\Lib\site-packages\torch\autograd\__init__.py:354, in
backward(tensors, grad_tensors, retain_graph, create_graph, grad_variables,
inputs)
     349     retain_graph = create_graph
     351 # The reason we repeat the same comment below is that
     352 # some Python versions print out the first line of a multi-line functio
     353 # calls in the traceback and some print out the last line
--> 354 _engine_run_backward(
     355     tensors,
     356     grad_tensors_,
     357     retain_graph,
     358     create_graph,
     359     inputs_tuple,
     360     allow_unreachable=True,
     361     accumulate_grad=True,

```

```
362 )
```

```
File ~\anaconda3\Lib\site-packages\torch\autograd\graph.py:841, in
↳ _engine_run_backward(t_outputs, *args, **kwargs)
    839     unregister_hooks = _register_logging_hooks_on_whole_graph(t_outputs
    840 try:
--> 841     return Variable._execution_engine.run_backward( # Calls into the
↳ C++ engine to run the backward pass
    842         t_outputs, *args, **kwargs
    843     ) # Calls into the C++ engine to run the backward pass
    844 finally:
    845     if attach_logging_hooks:
```

KeyboardInterrupt:

```
[ ]: torch.save(model.state_dict(), "efficientnet_b0_model.pth")
```

```
[136]: from torchvision.models import resnet18, ResNet18_Weights

weights = ResNet18_Weights.DEFAULT
model = resnet18(weights=weights)

for param in model.layer1.parameters():
    param.requires_grad = False
for param in model.layer2.parameters():
    param.requires_grad = False
for param in model.layer3.parameters():
    param.requires_grad = False
for param in model.layer4.parameters():
    param.requires_grad = False

in_features = model.fc.in_features
model.fc = nn.Linear(in_features, num_classes)

model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=learning_rate)
```

```
[ ]: torch.save(model.state_dict(), "resnet18_model.pth")
```

```
[133]: from torchvision.models import densenet121, DenseNet121_Weights

weights = DenseNet121_Weights.DEFAULT
model = densenet121(weights=weights)

for param in model.features.parameters():
```

```

        param.requires_grad = False

    in_features = model.classifier.in_features
    model.classifier = nn.Linear(in_features, num_classes)

    model = model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.classifier.parameters(), lr=learning_rate)

```

Downloading: "https://download.pytorch.org/models/densenet121-a639ec97.pth" to
 C:\Users\Mohammed Al Malki/.cache\torch\hub\checkpoints\densenet121-a639ec97.pth
 100%| | 30.8M/30.8M [00:05<00:00, 6.25MB/s]

```

[ ]: import os
import torch
from torch import nn, optim
from torch.utils.data import DataLoader, Subset
from torchvision import datasets, transforms
from torchvision.models import vgg16, VGG16_Weights

# Data configuration
data_dir = "newdataset"
batch_size = 32
num_epochs = 10
learning_rate = 1e-4
val_ratio = 0.2

# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Data transforms
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.
↪02),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

val_transform = transforms.Compose([

```

```

        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])

# Load dataset and create train/validation splits
try:
    base_dataset = datasets.ImageFolder(data_dir)
    num_classes = len(base_dataset.classes)

    print(f"Dataset loaded successfully!")
    print(f"Number of classes: {num_classes}")
    print(f"Classes: {base_dataset.classes}")
    print(f"Total samples: {len(base_dataset)}")

    # Create train/validation splits
    num_samples = len(base_dataset)
    val_size = int(num_samples * val_ratio)
    train_size = num_samples - val_size

    # Create indices for splitting
    indices = torch.randperm(num_samples).tolist()
    train_indices = indices[:train_size]
    val_indices = indices[train_size:]

    print(f"Training samples: {train_size}")
    print(f"Validation samples: {val_size}")

    # Create datasets with transforms
    train_dataset_full = datasets.ImageFolder(data_dir,
    ↪transform=train_transform)
    val_dataset_full = datasets.ImageFolder(data_dir, transform=val_transform)

    train_dataset = Subset(train_dataset_full, train_indices)
    val_dataset = Subset(val_dataset_full, val_indices)

    # Create data loaders
    train_loader = DataLoader(
        train_dataset,
        batch_size=batch_size,
        shuffle=True,
        num_workers=2,
        pin_memory=True if torch.cuda.is_available() else False
    )

```

```

val_loader = DataLoader(
    val_dataset,
    batch_size=batch_size,
    shuffle=False,
    num_workers=2,
    pin_memory=True if torch.cuda.is_available() else False
)

except Exception as e:
    print(f"Error loading dataset: {e}")
    exit()

# Model configuration
try:
    weights = VGG16_Weights.DEFAULT
    model = vgg16(weights=weights)

    # Freeze feature extraction layers
    for param in model.features.parameters():
        param.requires_grad = False

    # Modify classifier for our number of classes
    in_features = model.classifier[6].in_features
    model.classifier[6] = nn.Linear(in_features, num_classes)

    model = model.to(device)
    print("Model configured successfully!")

except Exception as e:
    print(f"Error configuring model: {e}")
    exit()

# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
# Optimize only the classifier parameters (unfrozen ones)
optimizer = optim.Adam(
    filter(lambda p: p.requires_grad, model.parameters()),
    lr=learning_rate
)

# Training loop
print("Starting training...")
print("-" * 60)

for epoch in range(num_epochs):
    # Training phase
    model.train()

```

```

running_loss = 0.0
train_correct = 0
train_total = 0

for images, labels in train_loader:
    images = images.to(device, non_blocking=True)
    labels = labels.to(device, non_blocking=True)

    optimizer.zero_grad()
    outputs = model(images)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()

    running_loss += loss.item() * images.size(0)

    # Calculate training accuracy
    _, preds = torch.max(outputs, 1)
    train_total += labels.size(0)
    train_correct += (preds == labels).sum().item()

epoch_loss = running_loss / len(train_loader.dataset)
train_acc = 100.0 * train_correct / train_total

# Validation phase
model.eval()
val_correct = 0
val_total = 0
val_loss = 0.0

with torch.no_grad():
    for images, labels in val_loader:
        images = images.to(device, non_blocking=True)
        labels = labels.to(device, non_blocking=True)

        outputs = model(images)
        loss = criterion(outputs, labels)
        val_loss += loss.item() * images.size(0)

        _, preds = torch.max(outputs, 1)
        val_total += labels.size(0)
        val_correct += (preds == labels).sum().item()

val_loss = val_loss / len(val_loader.dataset)
val_acc = 100.0 * val_correct / val_total

print(f"Epoch [{epoch+1:02d}/{num_epochs}]")

```

```

print(f"  Train Loss: {epoch_loss:.4f}, Train Acc: {train_acc:.2f}%")
print(f"  Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%")
print("-" * 50)

# Final results
print("\nTraining completed!")
print(f"Final Validation Accuracy: {val_acc:.2f}%")
print("Classes:", base_dataset.classes)

# Save model
try:
    torch.save({
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'class_names': base_dataset.classes,
        'num_classes': num_classes,
        'epoch': num_epochs
    }, "vgg16_model.pth")
    print("Model saved successfully as vgg16_model.pth")

except Exception as e:
    print(f"Error saving model: {e}")

```

```

Using device: cpu
Dataset loaded successfully!
Number of classes: 4
Classes: ['cataract', 'diabetic_retinopathy', 'glaucoma', 'normal']
Total samples: 4217
Training samples: 3374
Validation samples: 843
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to
C:\Users\Mohammed Al Malki\.cache\torch\hub\checkpoints\vgg16-397923af.pth
100%|      | 528M/528M [01:00<00:00, 9.13MB/s]

Model configured successfully!
Starting training...
-----

```

[]:

[]: