

Project 01

SID: 253000264

Download Data from Kaggle

Download the **Eye Diseases Classification** dataset from:

<https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-classification/data>

After extracting the dataset, place the four class folders inside a folder named `dataset` :

```
dataset/
├── cataract/
├── diabetic_retinopathy/
├── glaucoma/
└── normal/
```

Import Libraries

```
In [1]: import os
import shutil
import numpy as np
import torch
from PIL import Image
from torchvision import datasets, transforms as T, models
from torch import nn, optim
from torch.utils.data import DataLoader
import warnings
warnings.filterwarnings("ignore")
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report
)
import matplotlib.pyplot as plt
import seaborn as sns
```

Reduce Image Size

```
In [2]: input_root = 'dataset/'
output_root = 'output/'
resize_size = (256, 256)

os.makedirs(output_root, exist_ok=True)

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

    output_class_path = os.path.join(output_root, class_name)
    os.makedirs(output_class_path, exist_ok=True)

    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:
                with Image.open(input_image_path) as img:
                    img = img.convert('RGB')
                    img = img.resize(resize_size)
                    img.save(output_image_path, 'JPEG')
            except Exception as e:
                print(f"Failed to process {input_image_path}: {e}")

print('Image resizing completed.')
```

Image resizing completed.

Splitting Dataset

```
In [3]: original_dataset_dir = 'output'
base_dir = 'new dataset'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
```

```

test_dir = os.path.join(base_dir, 'test')

for directory in [train_dir, validation_dir, test_dir]:
    os.makedirs(directory, exist_ok=True)

train_ratio = 0.7
validation_ratio = 0.1
test_ratio = 0.2

def split_and_copy_files(class_dir, dest_base_dir):
    all_images = [
        os.path.join(class_dir, file_name)
        for file_name in os.listdir(class_dir)
        if os.path.isfile(os.path.join(class_dir, file_name))
    ]

    train_images, temp_images = train_test_split(
        all_images,
        test_size=(1 - train_ratio),
        random_state=42
    )

    validation_images, test_images = train_test_split(
        temp_images,
        test_size=test_ratio / (validation_ratio + test_ratio),
        random_state=42
    )

    for images, folder_name in zip(
        [train_images, validation_images, test_images],
        ['train', 'validation', 'test']
    ):
        destination = os.path.join(
            dest_base_dir,
            folder_name,
            os.path.basename(class_dir)
        )
        os.makedirs(destination, exist_ok=True)

        for file_path in images:
            shutil.copy(file_path, destination)

for class_name in os.listdir(original_dataset_dir):
    full_class_dir = os.path.join(original_dataset_dir, class_name)

    if os.path.isdir(full_class_dir):
        split_and_copy_files(full_class_dir, base_dir)

print('Dataset splitting completed.')

```

Dataset splitting completed.

Getting Name of Classes in Dataset

```

In [4]: def get_classes(data_dir):
        all_data = datasets.ImageFolder(data_dir)
        return all_data.classes

classes = get_classes('new dataset/test')
print(classes, len(classes))

```

['cataract', 'diabetic_retinopathy', 'glaucoma', 'normal'] 4

Image Augmentation and Preprocessing

```

In [5]: def get_data_loaders(data_dir, batch_size, train=False):
        """Prepare data loaders for training, validation, and testing."""

        if train:
            transform = T.Compose([
                T.RandomHorizontalFlip(),
                T.RandomVerticalFlip(),
                T.RandomRotation(10),
                T.Resize((299, 299)),
                T.ToTensor(),
                T.Normalize(
                    mean=[0.485, 0.456, 0.406],
                    std=[0.229, 0.224, 0.225]
                )
            ])

            train_data = datasets.ImageFolder(
                os.path.join(data_dir, 'train/'),
                transform=transform
            )

            train_loader = DataLoader(
                train_data,
                batch_size=batch_size,
                shuffle=True,

```

```

        num_workers=0
    )

    return train_loader, len(train_data)

else:
    transform = T.Compose([
        T.Resize((299, 299)),
        T.ToTensor(),
        T.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])

    validation_data = datasets.ImageFolder(
        os.path.join(data_dir, 'validation/'),
        transform=transform
    )

    test_data = datasets.ImageFolder(
        os.path.join(data_dir, 'test/'),
        transform=transform
    )

    validation_loader = DataLoader(
        validation_data,
        batch_size=batch_size,
        shuffle=False,
        num_workers=0
    )

    test_loader = DataLoader(
        test_data,
        batch_size=batch_size,
        shuffle=False,
        num_workers=0
    )

    return (
        validation_loader,
        test_loader,
        len(validation_data),
        len(test_data)
    )

```

```

In [6]: dataset_path = 'new dataset'
batch_size = 32

train_loader, train_data_len = get_data_loaders(
    dataset_path,
    batch_size,
    train=True
)

validation_loader, test_loader, validation_data_len, test_data_len = get_data_loaders(
    dataset_path,
    batch_size,
    train=False
)

classes = get_classes('new dataset/train')
num_classes = len(classes)

dataloaders = {
    'train': train_loader,
    'val': validation_loader
}

dataset_sizes = {
    'train': train_data_len,
    'val': validation_data_len
}

if torch.cuda.is_available():
    device = torch.device('cuda')
elif torch.backends.mps.is_available():
    device = torch.device('mps')
else:
    device = torch.device('cpu')
print('Device:', device)
print('Classes:', classes)

```

Device: mps
Classes: ['cataract', 'diabetic_retinopathy', 'glaucoma', 'normal']

Training Function

```

In [7]: def train_model(model, criterion, optimizer, scheduler, num_epochs=20):
        """Train and validate the model."""

        for epoch in range(num_epochs):
            print(f"Epoch {epoch + 1}/{num_epochs}")

```

```

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)

            if hasattr(outputs, 'logits'):
                main_outputs = outputs.logits
                auxiliary_outputs = getattr(outputs, 'aux_logits', None)
            elif isinstance(outputs, tuple):
                main_outputs = outputs[0]
                auxiliary_outputs = outputs[1] if len(outputs) > 1 else None
            else:
                main_outputs = outputs
                auxiliary_outputs = None

            loss = criterion(main_outputs, labels)

            if phase == 'train' and auxiliary_outputs is not None:
                loss += 0.4 * criterion(auxiliary_outputs, labels)

            _, predictions = torch.max(main_outputs, 1)

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

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

    if phase == 'train':
        scheduler.step()

    epoch_loss = running_loss / dataset_sizes[phase]
    epoch_accuracy = running_corrects.float() / dataset_sizes[phase]

    print(
        f"{phase} Loss: {epoch_loss:.4f} "
        f"Acc: {epoch_accuracy:.4f}"
    )

return model

```

Evaluation Function

```

In [8]: def evaluate_model(model_name, model, test_loader, classes):
        y_true = []
        y_pred = []

        model.eval()

        with torch.no_grad():
            for inputs, labels in tqdm(test_loader):
                outputs = model(inputs.to(device))

                if hasattr(outputs, 'logits'):
                    outputs = outputs.logits
                elif isinstance(outputs, tuple):
                    outputs = outputs[0]

                probabilities = torch.softmax(outputs, dim=1)
                predictions = torch.argmax(probabilities, dim=1).cpu().numpy()

                y_true.extend(labels.numpy())
                y_pred.extend(predictions)

        y_true = np.array(y_true)
        y_pred = np.array(y_pred)

        accuracy = accuracy_score(y_true, y_pred)
        precision = precision_score(y_true, y_pred, average='weighted', zero_division=0)
        recall = recall_score(y_true, y_pred, average='weighted', zero_division=0)
        f1 = f1_score(y_true, y_pred, average='weighted', zero_division=0)
        matrix = confusion_matrix(y_true, y_pred)

        print('Model:', model_name)
        print('Accuracy:', accuracy)
        print('Precision:', precision)
        print('Recall:', recall)

```

```
print('F1-score:', f1)

print('\nClassification Report:')
print(
    classification_report(
        y_true,
        y_pred,
        target_names=classes,
        zero_division=0
    )
)

plt.figure(figsize=(6, 5))
sns.heatmap(
    matrix,
    annot=True,
    fmt='d',
    cmap='Blues',
    xticklabels=classes,
    yticklabels=classes
)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title(f'{model_name} Confusion Matrix')
plt.show()
```

EfficientNetB0

```
In [9]: model = models.efficientnet_b0(pretrained=True)
model.classifier[1] = nn.Linear(
    model.classifier[1].in_features,
    num_classes
)
model = model.to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(
    optimizer,
    step_size=3,
    gamma=0.97
)

model = train_model(
    model,
    criterion,
    optimizer,
    exp_lr_scheduler,
    num_epochs=2
)

evaluate_model(
    'efficientnet_b0',
    model,
    test_loader,
    classes
)
```

Epoch 1/2

train Loss: 0.5431 Acc: 0.7976
val Loss: 0.3048 Acc: 0.8839

Epoch 2/2

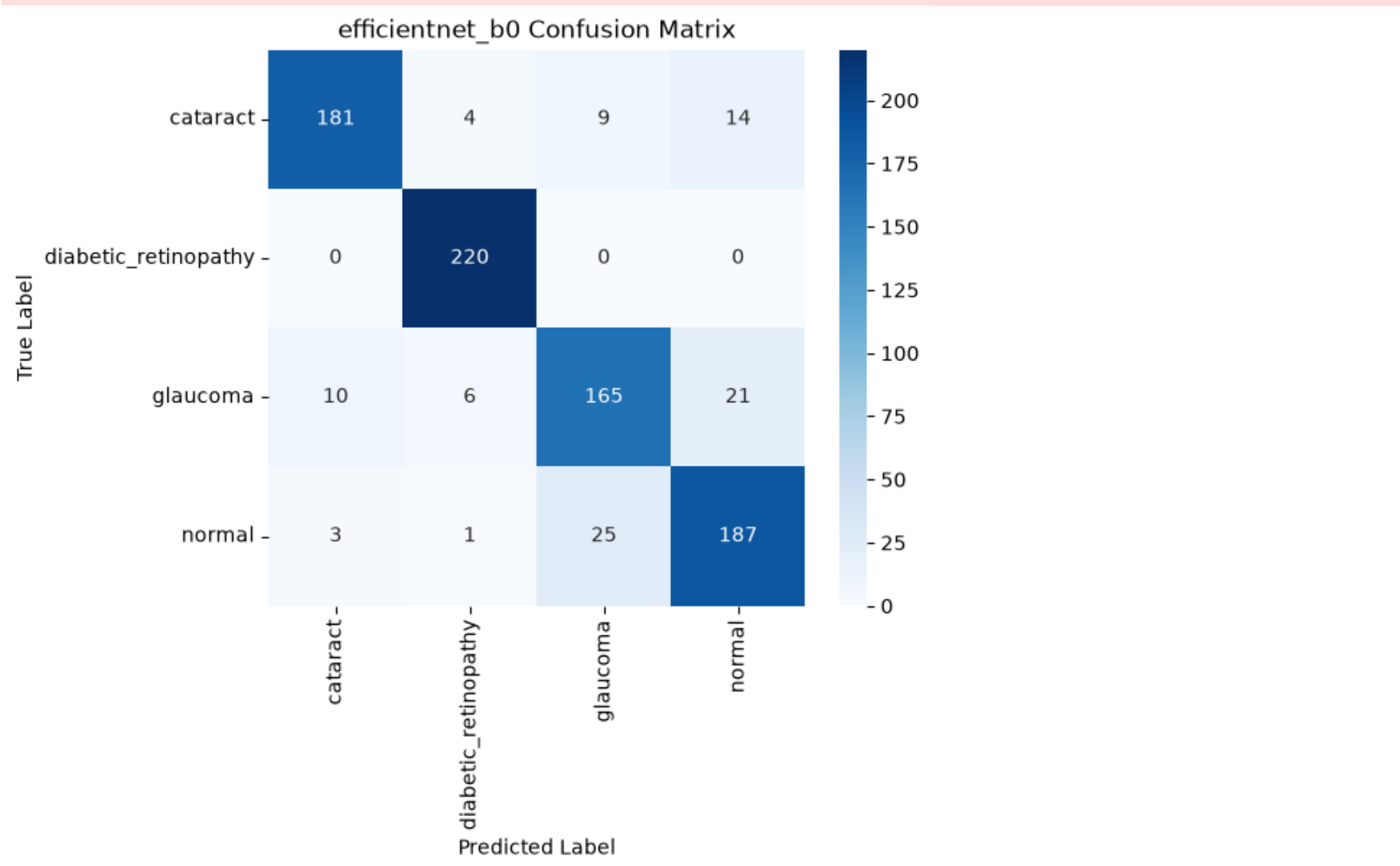
train Loss: 0.3565 Acc: 0.8732
val Loss: 0.2940 Acc: 0.9005

0%		0/27	[00:00<?, ?it/s]
4%		1/27	[00:00<00:07, 3.68it/s]
7% █		2/27	[00:00<00:06, 3.77it/s]
11% █		3/27	[00:00<00:06, 3.82it/s]
15% █		4/27	[00:01<00:05, 3.86it/s]
19% █		5/27	[00:01<00:05, 3.88it/s]
22% █		6/27	[00:01<00:05, 3.88it/s]
26% █		7/27	[00:01<00:05, 3.88it/s]
30% █		8/27	[00:02<00:04, 3.87it/s]
33% █		9/27	[00:02<00:04, 3.86it/s]
37% █		10/27	[00:02<00:04, 3.86it/s]
41% █		11/27	[00:02<00:04, 3.87it/s]
44% █		12/27	[00:03<00:03, 3.88it/s]
48% █		13/27	[00:03<00:03, 3.78it/s]
52% █		14/27	[00:03<00:03, 3.51it/s]
56% █		15/27	[00:03<00:03, 3.62it/s]
59% █		16/27	[00:04<00:02, 3.71it/s]
63% █		17/27	[00:04<00:02, 3.77it/s]
67% █		18/27	[00:04<00:02, 3.77it/s]
70% █		19/27	[00:05<00:02, 3.77it/s]
74% █		20/27	[00:05<00:01, 3.79it/s]

78%			21/27	[00:05<00:01,	3.80it/s]
81%			22/27	[00:05<00:01,	3.83it/s]
85%			23/27	[00:06<00:01,	3.85it/s]
89%			24/27	[00:06<00:00,	3.87it/s]
93%			25/27	[00:06<00:00,	3.86it/s]
96%			26/27	[00:06<00:00,	3.86it/s]
100%			27/27	[00:07<00:00,	3.89it/s]
100%			27/27	[00:07<00:00,	3.81it/s]

Model: efficientnet_b0
Accuracy: 0.8900709219858156
Precision: 0.8900934377373384
Recall: 0.8900709219858156
F1-score: 0.8896104007777764

Classification Report:					
		precision	recall	f1-score	support
diabetic_retinopathy	cataract	0.93	0.87	0.90	208
	diabetic_retinopathy	0.95	1.00	0.98	220
	glaucoma	0.83	0.82	0.82	202
	normal	0.84	0.87	0.85	216
accuracy				0.89	846
macro avg		0.89	0.89	0.89	846
weighted avg		0.89	0.89	0.89	846



ResNet18

```
In [10]: model = models.resnet18(pretrained=False)
model.fc = nn.Linear(model.fc.in_features, num_classes)
model = model.to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(
    optimizer,
    step_size=3,
    gamma=0.97
)

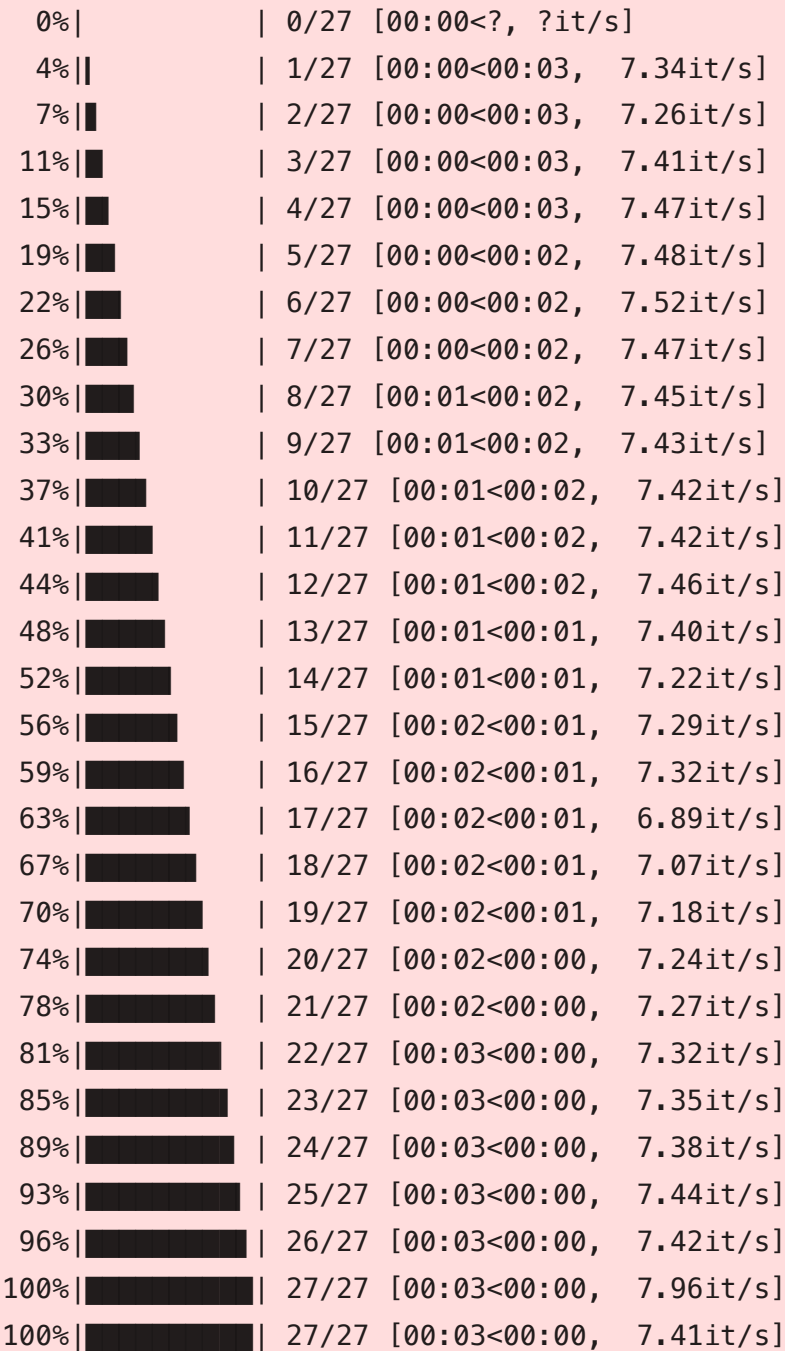
model = train_model(
    model,
    criterion,
    optimizer,
    exp_lr_scheduler,
    num_epochs=2
)

evaluate_model(
    'ResNet18',
    model,
    test_loader,
    classes
)
```

Epoch 1/2

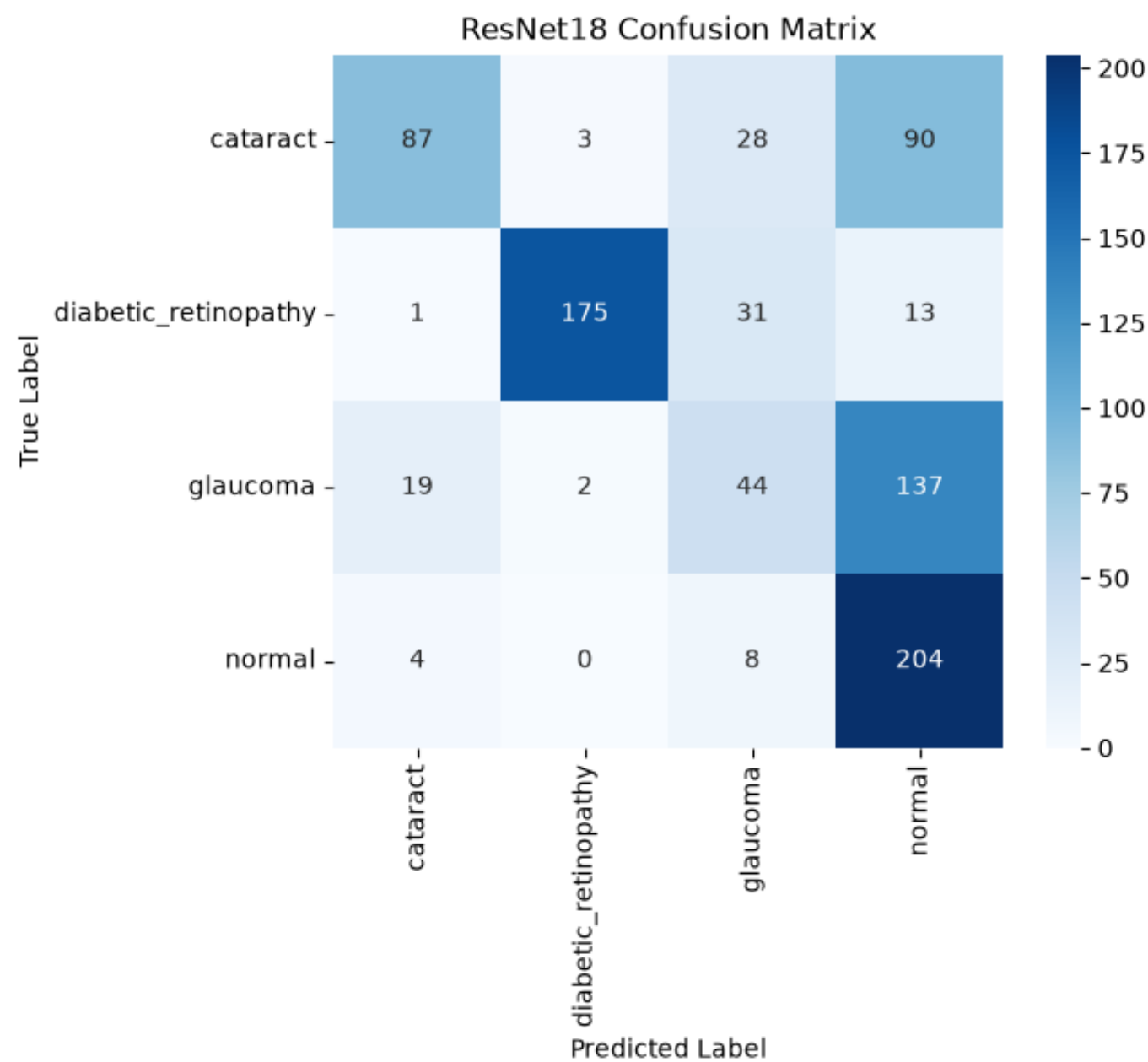
train Loss: 1.1634 Acc: 0.4927
val Loss: 1.0418 Acc: 0.5877
Epoch 2/2

train Loss: 0.8980 Acc: 0.5992
val Loss: 0.8618 Acc: 0.6185



Model: ResNet18
Accuracy: 0.6028368794326241
Precision: 0.6574837248596113
Recall: 0.6028368794326241
F1-score: 0.5866124709813106

Classification Report:					
	precision	recall	f1-score	support	
cataract	0.78	0.42	0.55	208	
diabetic_retinopathy	0.97	0.80	0.88	220	
glaucoma	0.40	0.22	0.28	202	
normal	0.46	0.94	0.62	216	
accuracy			0.60	846	
macro avg	0.65	0.59	0.58	846	
weighted avg	0.66	0.60	0.59	846	



DenseNet121

```
In [11]: model = models.densenet121(pretrained=False)
model.classifier = nn.Linear(
    model.classifier.in_features,
    num_classes
)
model = model.to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(
    optimizer,
    step_size=3,
    gamma=0.97
)

model = train_model(
    model,
    criterion,
    optimizer,
    exp_lr_scheduler,
    num_epochs=1
)

evaluate_model(
    'DenseNet121',
    model,
    test_loader,
    classes
)
```

Epoch 1/1

train Loss: 1.1908 Acc: 0.4649
val Loss: 1.1242 Acc: 0.5142

0%	0/27 [00:00<?, ?it/s]
4%	1/27 [00:00<00:09, 2.85it/s]
7%	2/27 [00:00<00:08, 2.87it/s]
11%	3/27 [00:01<00:08, 2.93it/s]
15%	4/27 [00:01<00:07, 2.96it/s]
19%	5/27 [00:01<00:07, 2.98it/s]
22%	6/27 [00:02<00:07, 3.00it/s]
26%	7/27 [00:02<00:06, 3.00it/s]
30%	8/27 [00:02<00:06, 2.97it/s]
33%	9/27 [00:03<00:06, 2.97it/s]
37%	10/27 [00:03<00:05, 2.97it/s]
41%	11/27 [00:03<00:05, 2.97it/s]
44%	12/27 [00:04<00:05, 2.97it/s]
48%	13/27 [00:04<00:04, 2.97it/s]
52%	14/27 [00:04<00:04, 2.96it/s]
56%	15/27 [00:05<00:04, 2.95it/s]

59%

63%

67%

70%

74%

78%

81%

85%

89%

93%

96%

100%

100%

16/27

[00:05<00:03,

2.95it/s]

17/27

[00:05<00:03,

2.95it/s]

18/27

[00:06<00:03,

2.96it/s]

19/27

[00:06<00:02,

2.96it/s]

20/27

[00:06<00:02,

2.95it/s]

21/27

[00:07<00:02,

2.94it/s]

22/27

[00:07<00:01,

2.94it/s]

23/27

[00:07<00:01,

2.95it/s]

24/27

[00:08<00:01,

2.96it/s]

25/27

[00:08<00:00,

2.96it/s]

26/27

[00:08<00:00,

2.96it/s]

27/27

[00:09<00:00,

2.25it/s]

27/27

[00:09<00:00,

2.85it/s]

Model: DenseNet121

Accuracy: 0.46217494089834515

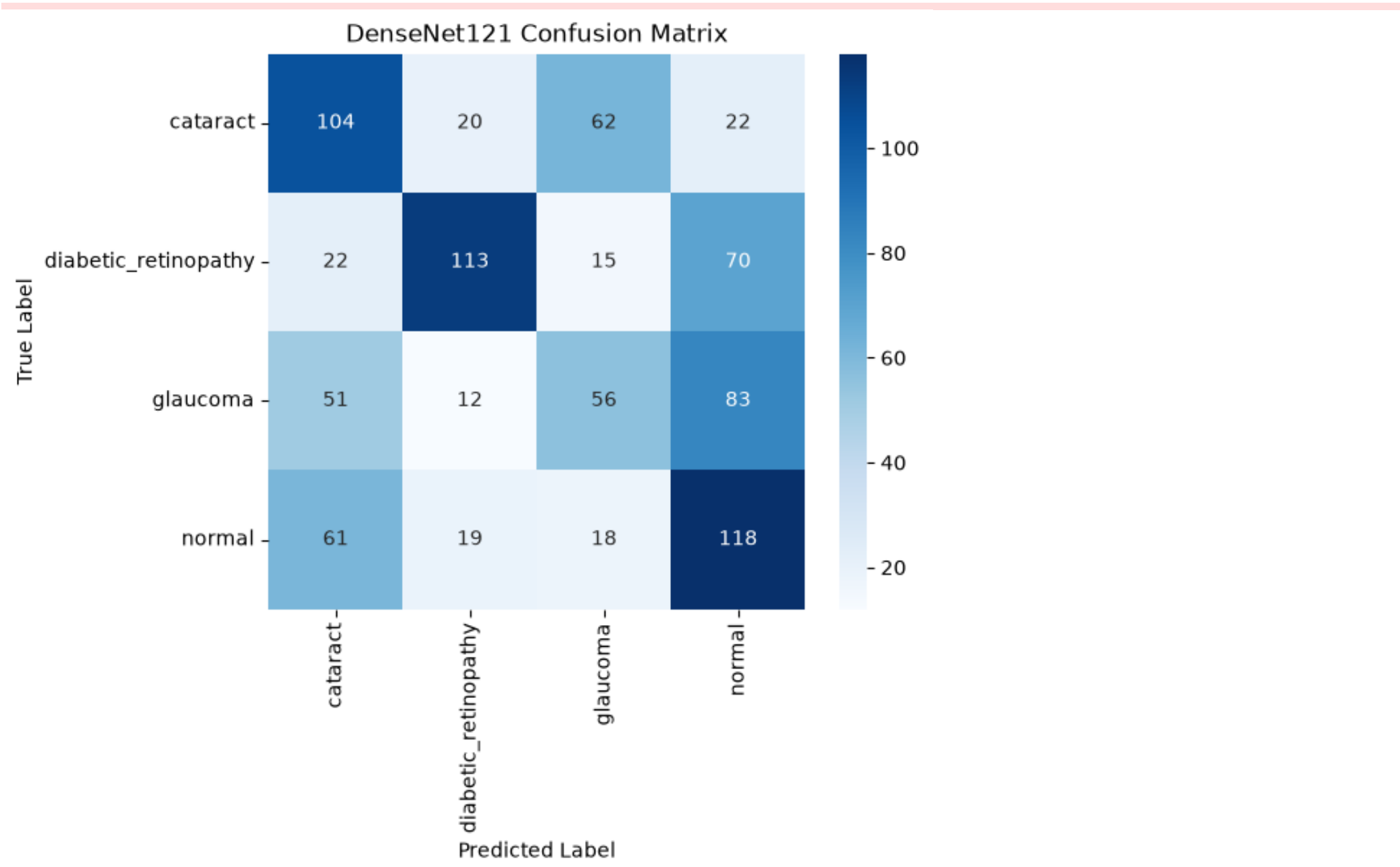
Precision: 0.4779902961384136

Recall: 0.46217494089834515

F1-score: 0.461848244570784

Classification Report:

	precision	recall	f1-score	support
cataract	0.44	0.50	0.47	208
diabetic_retinopathy	0.69	0.51	0.59	220
glaucoma	0.37	0.28	0.32	202
normal	0.40	0.55	0.46	216
accuracy			0.46	846
macro avg	0.47	0.46	0.46	846
weighted avg	0.48	0.46	0.46	846



VGG16

```
In [12]: _original_device = device
if device.type == 'mps':
    # VGG16's AdaptiveAvgPool2d((7,7)) on a 299x299 input produces a 9x9
    # feature map, which MPS cannot adaptively pool (non-divisible sizes
    # are unsupported on this backend) -> fall back to CPU for this model.
    device = torch.device('cpu')

model = models.vgg16(pretrained=False)
model.classifier[6] = nn.Linear(
    model.classifier[6].in_features,
    num_classes
)
model = model.to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(
    optimizer,
    step_size=3,
```

```
        gamma=0.97
    )

    model = train_model(
        model,
        criterion,
        optimizer,
        exp_lr_scheduler,
        num_epochs=1
    )

    evaluate_model(
        'VGG16',
        model,
        test_loader,
        classes
    )

    device = _original_device
```

Epoch 1/1

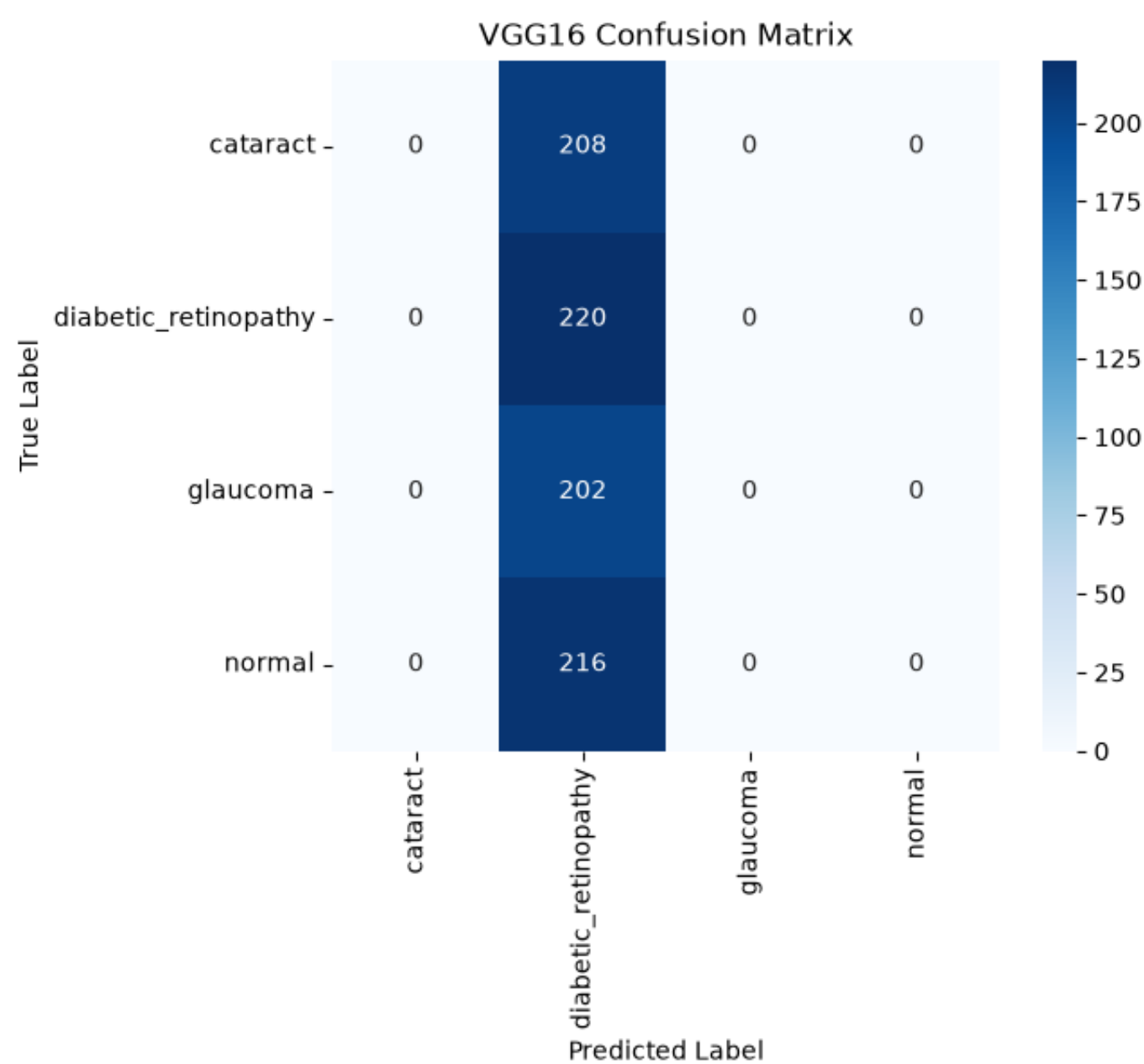
train Loss: 4.5792 Acc: 0.2564
val Loss: 1.3911 Acc: 0.2607

0%		0/27	[00:00<?, ?it/s]
4%		1/27	[00:03<01:42, 3.95s/it]
7%		2/27	[00:07<01:39, 3.96s/it]
11%		3/27	[00:11<01:35, 3.96s/it]
15%		4/27	[00:15<01:31, 3.97s/it]
19%		5/27	[00:19<01:27, 3.97s/it]
22%		6/27	[00:23<01:23, 3.98s/it]
26%		7/27	[00:27<01:19, 3.98s/it]
30%		8/27	[00:31<01:15, 3.98s/it]
33%		9/27	[00:35<01:11, 3.97s/it]
37%		10/27	[00:39<01:07, 3.97s/it]
41%		11/27	[00:43<01:03, 3.98s/it]
44%		12/27	[00:47<00:59, 3.97s/it]
48%		13/27	[00:51<00:55, 3.97s/it]
52%		14/27	[00:55<00:51, 3.97s/it]
56%		15/27	[00:59<00:47, 3.97s/it]
59%		16/27	[01:03<00:43, 3.97s/it]
63%		17/27	[01:07<00:39, 3.97s/it]
67%		18/27	[01:11<00:35, 3.96s/it]
70%		19/27	[01:15<00:31, 3.96s/it]
74%		20/27	[01:19<00:27, 3.96s/it]
78%		21/27	[01:23<00:23, 3.96s/it]
81%		22/27	[01:27<00:19, 3.96s/it]
85%		23/27	[01:31<00:15, 3.97s/it]
89%		24/27	[01:35<00:11, 3.97s/it]
93%		25/27	[01:39<00:07, 3.98s/it]
96%		26/27	[01:43<00:03, 4.00s/it]
100%		27/27	[01:45<00:00, 3.51s/it]
100%		27/27	[01:45<00:00, 3.91s/it]

Model: VGG16
Accuracy: 0.26004728132387706
Precision: 0.06762458852393965
Recall: 0.26004728132387706
F1-score: 0.10733658891417064

Classification Report:

	precision	recall	f1-score	support
cataract	0.00	0.00	0.00	208
diabetic_retinopathy	0.26	1.00	0.41	220
glaucoma	0.00	0.00	0.00	202
normal	0.00	0.00	0.00	216
accuracy			0.26	846
macro avg	0.07	0.25	0.10	846
weighted avg	0.07	0.26	0.11	846



InceptionV3

```
In [13]: model = models.inception_v3(
    pretrained=False,
    aux_logits=True
)
model.fc = nn.Linear(model.fc.in_features, num_classes)

if model.aux_logits:
    model.AuxLogits.fc = nn.Linear(
        model.AuxLogits.fc.in_features,
        num_classes
    )

model = model.to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(
    optimizer,
    step_size=3,
    gamma=0.97
)

model = train_model(
    model,
    criterion,
    optimizer,
    exp_lr_scheduler,
    num_epochs=1
)

evaluate_model(
    'InceptionV3',
    model,
    test_loader,
    classes
)
```

Epoch 1/1

train Loss: 1.7199 Acc: 0.4557

val Loss: 1.0231 Acc: 0.5782

```
0%|          | 0/27 [00:00<?, ?it/s]
4%||         | 1/27 [00:00<00:07, 3.28it/s]
7%|||        | 2/27 [00:00<00:07, 3.44it/s]
11%|||       | 3/27 [00:00<00:06, 3.55it/s]
15%||||      | 4/27 [00:01<00:06, 3.62it/s]
19%|||||     | 5/27 [00:01<00:06, 3.66it/s]
22%|||||     | 6/27 [00:01<00:05, 3.70it/s]
26%|||||     | 7/27 [00:01<00:05, 3.71it/s]
30%|||||     | 8/27 [00:02<00:05, 3.69it/s]
33%|||||     | 9/27 [00:02<00:04, 3.70it/s]
```

37%			10/27	[00:02<00:04,	3.67it/s]
41%			11/27	[00:03<00:04,	3.70it/s]
44%			12/27	[00:03<00:04,	3.71it/s]
48%			13/27	[00:03<00:03,	3.73it/s]
52%			14/27	[00:03<00:03,	3.72it/s]
56%			15/27	[00:04<00:03,	3.73it/s]
59%			16/27	[00:04<00:02,	3.73it/s]
63%			17/27	[00:04<00:02,	3.75it/s]
67%			18/27	[00:04<00:02,	3.75it/s]
70%			19/27	[00:05<00:02,	3.75it/s]
74%			20/27	[00:05<00:01,	3.73it/s]
78%			21/27	[00:05<00:01,	3.72it/s]
81%			22/27	[00:05<00:01,	3.73it/s]
85%			23/27	[00:06<00:01,	3.74it/s]
89%			24/27	[00:06<00:00,	3.71it/s]
93%			25/27	[00:06<00:00,	3.70it/s]
96%			26/27	[00:07<00:00,	3.72it/s]
100%			27/27	[00:07<00:00,	2.93it/s]
100%			27/27	[00:07<00:00,	3.58it/s]

Model: InceptionV3
Accuracy: 0.5602836879432624
Precision: 0.5499582998842424
Recall: 0.5602836879432624
F1-score: 0.5386240096148422

Classification Report:					
	precision	recall	f1-score	support	
cataract	0.64	0.77	0.70	208	
diabetic_retinopathy	0.63	0.61	0.62	220	
glaucoma	0.45	0.20	0.27	202	
normal	0.47	0.64	0.55	216	
accuracy			0.56	846	
macro avg	0.55	0.56	0.54	846	
weighted avg	0.55	0.56	0.54	846	

