

Project 1

✓ Eye Disease Classification

Student Name:

Waleed Ahmed AlSharif

Dataset:

Eye Diseases Classification Dataset (Kaggle)

```
import random
import numpy as np
import torch

seed = 42

random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)

if torch.cuda.is_available():
    torch.cuda.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)

torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

```
# =====
# Import Required Libraries
# =====

import os
import copy
import time
import random
import shutil
import warnings

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from PIL import Image

import torch
import torch.nn as nn
import torch.optim as optim

from torchvision import datasets
from torchvision import transforms
from torchvision import models

from torch.utils.data import DataLoader

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    classification_report
)

import seaborn as sns

warnings.filterwarnings("ignore")
```

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

print("Device:", device)
```

Device: cuda

```
def set_seed(seed=42):

    random.seed(seed)
    np.random.seed(seed)

    torch.manual_seed(seed)

    if torch.cuda.is_available():
        torch.cuda.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)

set_seed()
```

```
from google.colab import files
files.upload()
```

kaggle.json
kaggle.json(application/json) - 71 bytes, last modified: 7/12/2026 - 100% done
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"waleedalshareef","key":"766527770a3b2e65d50e45fefc46142e"}'}

```
import os
import shutil

os.makedirs('/root/.kaggle', exist_ok=True)
shutil.move('kaggle.json', '/root/.kaggle/kaggle.json')
os.chmod('/root/.kaggle/kaggle.json', 600)
```

```
!pip install -q kaggle
```

```
!kaggle datasets download -d gunavenkatdoddi/eye-diseases-classification
```

Warning: Looks like you're using an outdated `kaggle` version (installed: 2.0.2), please consider upgrading to the latest version
Dataset URL: <https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-classification>
License(s): ODbL-1.0
Downloading eye-diseases-classification.zip to /content
100% 736M/736M [00:25<00:00, 30.7MB/s]

```
!unzip -q eye-diseases-classification.zip
```

```
import os

os.listdir()

['.config', 'dataset', 'eye-diseases-classification.zip', 'sample_data']
```

```
for root, dirs, files in os.walk("."):
    print(root)
```

```

./
./config
./config/logs
./config/logs/2026.06.04
./config/configurations
./dataset
./dataset/diabetic_retinopathy
./dataset/cataract
./dataset/normal
./dataset/glaucoma
./sample_data
```

```
import os

dataset_path = "/content"

for root, dirs, files in os.walk(dataset_path):
    print(root)
```

```
/content
/content/.config
/content/.config/logs
/content/.config/logs/2026.06.04
/content/.config/configurations
/content/dataset
/content/dataset/diabetic_retinopathy
/content/dataset/cataract
```

```
/content/dataset/normal
/content/dataset/glaucoma
/content/sample_data
```

```
# =====
# Dataset Splitting
# =====
```

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

source_dir = "/content/dataset"
output_dir = "/content/eye_dataset"

classes = os.listdir(source_dir)

print("Classes:", classes)

for cls in classes:

    class_path = os.path.join(source_dir, cls)

    images = os.listdir(class_path)

    train_imgs, temp_imgs = train_test_split(
        images,
        test_size=0.30,
        random_state=42,
        shuffle=True
    )

    val_imgs, test_imgs = train_test_split(
        temp_imgs,
        test_size=0.50,
        random_state=42,
        shuffle=True
    )

    splits = {
        "train": train_imgs,
        "valid": val_imgs,
        "test": test_imgs
    }

    for split_name, split_images in splits.items():

        save_folder = os.path.join(output_dir, split_name, cls)

        os.makedirs(save_folder, exist_ok=True)

        for img in split_images:

            shutil.copy(
                os.path.join(class_path, img),
                os.path.join(save_folder, img)
            )

print("Dataset split completed successfully!")
```

```
Classes: ['diabetic_retinopathy', 'cataract', 'normal', 'glaucoma']
Dataset split completed successfully!
```

```
for split in ["train", "valid", "test"]:

    print(f"\n{split.upper()}")

    split_path = os.path.join(output_dir, split)

    for cls in classes:

        count = len(os.listdir(os.path.join(split_path, cls)))

        print(f"{cls}: {count}")
```

```
TRAIN
diabetic_retinopathy: 768
cataract: 726
normal: 751
glaucoma: 704

VALID
```

```
diabetic_retinopathy: 165
cataract: 156
normal: 161
glaucoma: 151
```

```
TEST
diabetic_retinopathy: 165
cataract: 156
normal: 162
glaucoma: 152
```

✓ Data Augmentation

Data augmentation was applied to improve the model's ability to generalize and reduce overfitting.

The following transformations were applied to the training dataset:

- Resize images to 224×224 pixels.
- Random Horizontal Flip.
- Random Rotation.
- Color Jitter.
- Convert images to Tensor.
- Normalize images using ImageNet mean and standard deviation.

For the validation and testing datasets, only resizing and normalization were applied.

```
# =====
# Data Augmentation
# =====

from torchvision import transforms

train_transform = transforms.Compose([

    transforms.Resize((224,224)),

    transforms.RandomHorizontalFlip(),

    transforms.RandomRotation(20),

    transforms.ColorJitter(
        brightness=0.2,
        contrast=0.2,
        saturation=0.2
    ),

    transforms.ToTensor(),

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

])

test_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]
    )

])
```

```
from torchvision import datasets

train_dataset = datasets.ImageFolder(
    "/content/eye_dataset/train",
    transform=train_transform
)

validation_dataset = datasets.ImageFolder(
    "/content/eye_dataset/valid",
    transform=test_transform
)
```

```
test_dataset = datasets.ImageFolder(
    "/content/eye_dataset/test",
    transform=test_transform
)
```

```
print(len(train_dataset))
print(len(validation_dataset))
print(len(test_dataset))
```

```
2949
633
635
```

```
classes = train_dataset.classes

print(classes)
```

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

```
from torch.utils.data import DataLoader
```

```
train_loader = DataLoader(
    train_dataset,
    batch_size=32,
    shuffle=True
)
```

```
validation_loader = DataLoader(
    validation_dataset,
    batch_size=32,
    shuffle=False
)
```

```
test_loader = DataLoader(
    test_dataset,
    batch_size=32,
    shuffle=False
)
```

```
# =====
# Build EfficientNet-B0 Model
# =====

import torch.nn as nn
from torchvision import models

num_classes = len(classes)

model = models.efficientnet_b0(pretrained=True)

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

model = model.to(device)

print(model)
```

Downloading: "https://download.pytorch.org/models/efficientnet_b0_rwightman-7f5810bc.pth" to /root/.cache/torch/hub/checkpoint
100%|██████████| 20.5M/20.5M [00:00<00:00, 82.4MB/s]

```
EfficientNet(
  (features): Sequential(
    (0): Conv2dNormActivation(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): SiLU(inplace=True)
    )
    (1): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
            (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(32, 8, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(8, 32, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
          )
        )
      )
    )
  )
)
```

```

        (scale_activation): Sigmoid()
    )
    (2): Conv2dNormActivation(
      (0): Conv2d(32, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (stochastic_depth): StochasticDepth(p=0.0, mode=row)
)
)
(2): Sequential(
  (0): MBConv(
    (block): Sequential(
      (0): Conv2dNormActivation(
        (0): Conv2d(16, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): SiLU(inplace=True)
      )
      (1): Conv2dNormActivation(
        (0): Conv2d(96, 96, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=96, bias=False)
        (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): SiLU(inplace=True)
      )
      (2): SqueezeExcitation(
        (avgpool): AdaptiveAvgPool2d(output_size=1)
        (fc1): Conv2d(96, 4, kernel_size=(1, 1), stride=(1, 1))
        (fc2): Conv2d(4, 96, kernel_size=(1, 1), stride=(1, 1))
        (activation): SiLU(inplace=True)
        (scale_activation): Sigmoid()
      )
      (3): Conv2dNormActivation(
        (0): Conv2d(96, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
  )
)
)

```

```

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.AdamW(
    model.parameters(),
    lr=0.001
)

scheduler = torch.optim.lr_scheduler.StepLR(
    optimizer,
    step_size=3,
    gamma=0.97
)

```

```
train_model()
```

```

-----
NameError                                Traceback (most recent call last)
/tmp/ipykernel_375/1430278874.py in <cell line: 0>()
----> 1 train_model()

NameError: name 'train_model' is not defined

```

Next steps: [Explain error](#)

```

def train_model(model, train_loader, validation_loader, criterion,
                optimizer, scheduler, num_epochs=30):

    train_losses = []
    valid_losses = []

    train_accuracies = []
    valid_accuracies = []

    best_accuracy = 0

    for epoch in range(num_epochs):

        print(f"Epoch {epoch+1}/{num_epochs}")

        model.train()

        running_loss = 0
        correct = 0
        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()

    _, predicted = outputs.max(1)

    total += labels.size(0)

    correct += predicted.eq(labels).sum().item()

train_loss = running_loss / len(train_loader)

train_accuracy = correct / total

train_losses.append(train_loss)

train_accuracies.append(train_accuracy)

#####

model.eval()

running_loss = 0
correct = 0
total = 0

with torch.no_grad():

    for images, labels in validation_loader:

        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)

        loss = criterion(outputs, labels)

        running_loss += loss.item()

        _, predicted = outputs.max(1)

        total += labels.size(0)

        correct += predicted.eq(labels).sum().item()

    valid_loss = running_loss / len(validation_loader)

    valid_accuracy = correct / total

    valid_losses.append(valid_loss)

    valid_accuracies.append(valid_accuracy)

    scheduler.step()

    print(f"Train Loss : {train_loss:.4f}")

    print(f"Valid Loss : {valid_loss:.4f}")

    print(f"Train Accuracy : {train_accuracy:.4f}")

    print(f"Valid Accuracy : {valid_accuracy:.4f}")

    print("-"*40)

return model, train_losses, valid_losses, train_accuracies, valid_accuracies

```

```

model, train_losses, valid_losses, train_acc, valid_acc = train_model(
    model,
    train_loader,
    validation_loader,
    criterion,

```

```
optimizer,  
scheduler,  
num_epochs=30  
)
```

```
Train Loss : 0.3233  
Valid Loss : 0.2593  
Train Accuracy : 0.8871  
Valid Accuracy : 0.9100  
-----
```

```
Epoch 22/30  
Train Loss : 0.3155  
Valid Loss : 0.2615  
Train Accuracy : 0.8918  
Valid Accuracy : 0.9084  
-----
```

```
Epoch 23/30  
Train Loss : 0.3194  
Valid Loss : 0.2602  
Train Accuracy : 0.8813  
Valid Accuracy : 0.9163  
-----
```

```
Epoch 24/30  
Train Loss : 0.3185  
Valid Loss : 0.2626  
Train Accuracy : 0.8857  
Valid Accuracy : 0.9084  
-----
```

```
Epoch 25/30  
Train Loss : 0.3229  
Valid Loss : 0.2610  
Train Accuracy : 0.8922  
Valid Accuracy : 0.9084  
-----
```

```
Epoch 26/30  
Train Loss : 0.3276  
Valid Loss : 0.2610  
Train Accuracy : 0.8783  
Valid Accuracy : 0.9115  
-----
```

```
Epoch 27/30  
Train Loss : 0.3002  
Valid Loss : 0.2721  
Train Accuracy : 0.8925  
Valid Accuracy : 0.9052  
-----
```

```
Epoch 28/30  
Train Loss : 0.3026  
Valid Loss : 0.2686  
Train Accuracy : 0.8911  
Valid Accuracy : 0.9052  
-----
```

```
Epoch 29/30  
Train Loss : 0.3501  
Valid Loss : 0.2628  
Train Accuracy : 0.8810  
Valid Accuracy : 0.9100  
-----
```

```
Epoch 30/30  
Train Loss : 0.3141  
Valid Loss : 0.2700  
Train Accuracy : 0.8884  
Valid Accuracy : 0.9068  
-----
```

```
import torch  
  
print("CUDA Available:", torch.cuda.is_available())  
  
if torch.cuda.is_available():  
    print("GPU:", torch.cuda.get_device_name(0))  
else:  
    print("Running on CPU")
```

```
CUDA Available: True  
GPU: Tesla T4
```

```
model.eval()  
  
y_true = []  
y_pred = []  
  
with torch.no_grad():  
    for images, labels in test_loader:
```

```

images = images.to(device)
labels = labels.to(device)

outputs = model(images)

_, predicted = torch.max(outputs, 1)

y_true.extend(labels.cpu().numpy())
y_pred.extend(predicted.cpu().numpy())

```

```

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

accuracy = accuracy_score(y_true, y_pred)

precision = precision_score(y_true, y_pred, average='weighted')

recall = recall_score(y_true, y_pred, average='weighted')

f1 = f1_score(y_true, y_pred, average='weighted')

print(f"Accuracy : {accuracy:.4f}")
print(f"Precision : {precision:.4f}")
print(f"Recall : {recall:.4f}")
print(f"F1 Score : {f1:.4f}")

```

```

Accuracy : 0.8976
Precision : 0.9030
Recall : 0.8976
F1 Score : 0.8946

```

```

from sklearn.metrics import classification_report

print(classification_report(
    y_true,
    y_pred,
    target_names=classes
))

```

	precision	recall	f1-score	support
cataract	0.90	0.96	0.93	156
diabetic_retinopathy	0.99	1.00	0.99	165
glaucoma	0.92	0.68	0.78	152
normal	0.80	0.94	0.86	162
accuracy			0.90	635
macro avg	0.90	0.89	0.89	635
weighted avg	0.90	0.90	0.89	635

```

import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns

cm = confusion_matrix(y_true, y_pred)

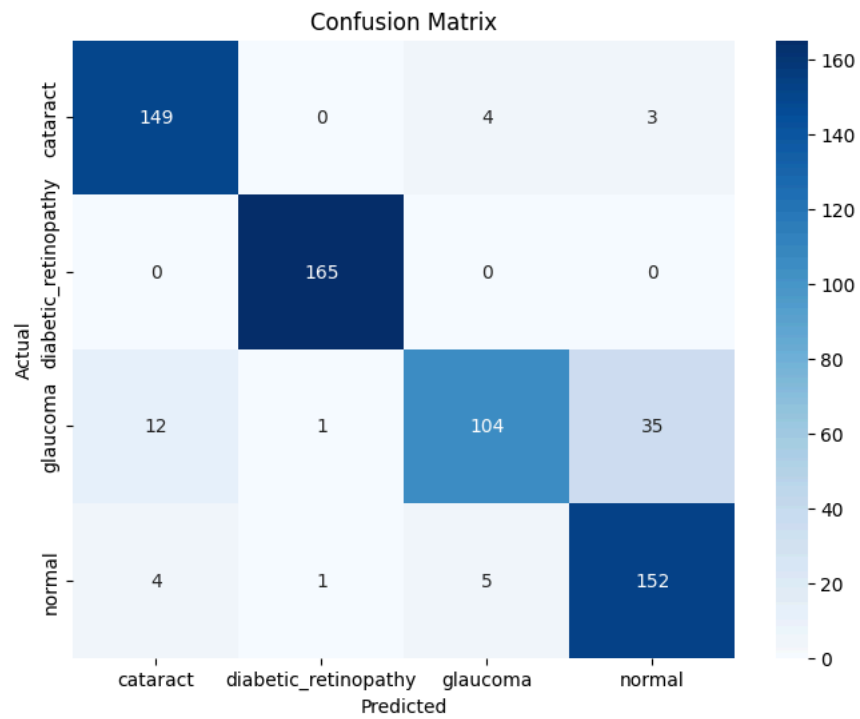
plt.figure(figsize=(8,6))

sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=classes,
    yticklabels=classes
)

plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")

plt.show()

```



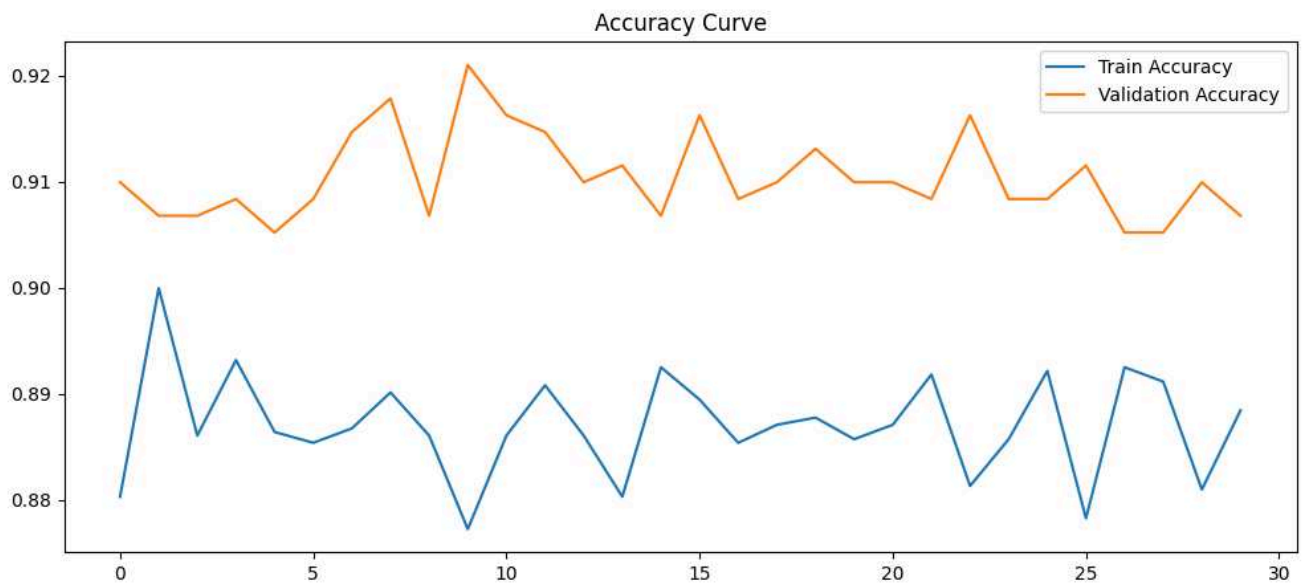
```
plt.figure(figsize=(12,5))

plt.plot(train_acc,label="Train Accuracy")
plt.plot(valid_acc,label="Validation Accuracy")

plt.legend()

plt.title("Accuracy Curve")

plt.show()
```



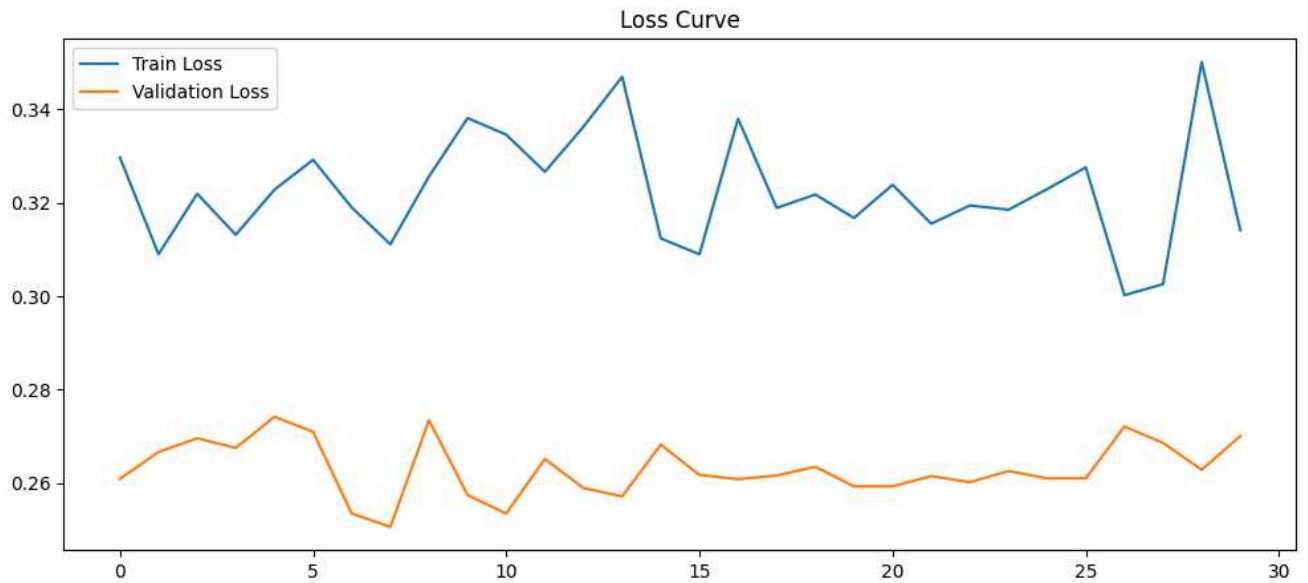
```
plt.figure(figsize=(12,5))

plt.plot(train_losses,label="Train Loss")
plt.plot(valid_losses,label="Validation Loss")

plt.legend()

plt.title("Loss Curve")

plt.show()
```



Start coding or [generate](#) with AI.

Start coding or [generate](#) with AI.

```
# =====  
# Model Evaluation  
# =====  
  
model.eval()  
  
y_true = []  
y_pred = []  
  
with torch.no_grad():  
    for images, labels in test_loader:  
        images = images.to(device)  
        labels = labels.to(device)  
  
        outputs = model(images)  
  
        _, predicted = torch.max(outputs, 1)  
  
        y_true.extend(labels.cpu().numpy())  
        y_pred.extend(predicted.cpu().numpy())  
  
print("Evaluation Completed Successfully!")
```

Evaluation Completed Successfully!

```
from sklearn.metrics import accuracy_score  
from sklearn.metrics import precision_score  
from sklearn.metrics import recall_score  
from sklearn.metrics import f1_score  
  
accuracy = accuracy_score(y_true, y_pred)  
  
precision = precision_score(  
    y_true,  
    y_pred,  
    average="weighted"  
)  
  
recall = recall_score(  
    y_true,  
    y_pred,  
    average="weighted"  
)  
  
f1 = f1_score(  
    y_true,  
    y_pred,  
    average="weighted"
```

```

)

print("="*40)
print(f"Accuracy : {accuracy:.4f}")
print(f"Precision : {precision:.4f}")
print(f"Recall : {recall:.4f}")
print(f"F1 Score : {f1:.4f}")
print("="*40)

```

```

=====
Accuracy : 0.2661
Precision : 0.2144
Recall : 0.2661
F1 Score : 0.1463
=====

```

```

from sklearn.metrics import classification_report

print(classification_report(
    y_true,
    y_pred,
    target_names=classes
))

```

	precision	recall	f1-score	support
cataract	0.96	0.97	0.97	156
diabetic_retinopathy	0.99	1.00	1.00	165
glaucoma	0.96	0.85	0.90	152
normal	0.88	0.95	0.91	162
accuracy			0.94	635
macro avg	0.95	0.94	0.94	635
weighted avg	0.95	0.94	0.94	635

```

from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(8,6))

plt.imshow(cm, cmap="Blues")

plt.title("Confusion Matrix")

plt.colorbar()

plt.xticks(range(len(classes)), classes, rotation=45)

plt.yticks(range(len(classes)), classes)

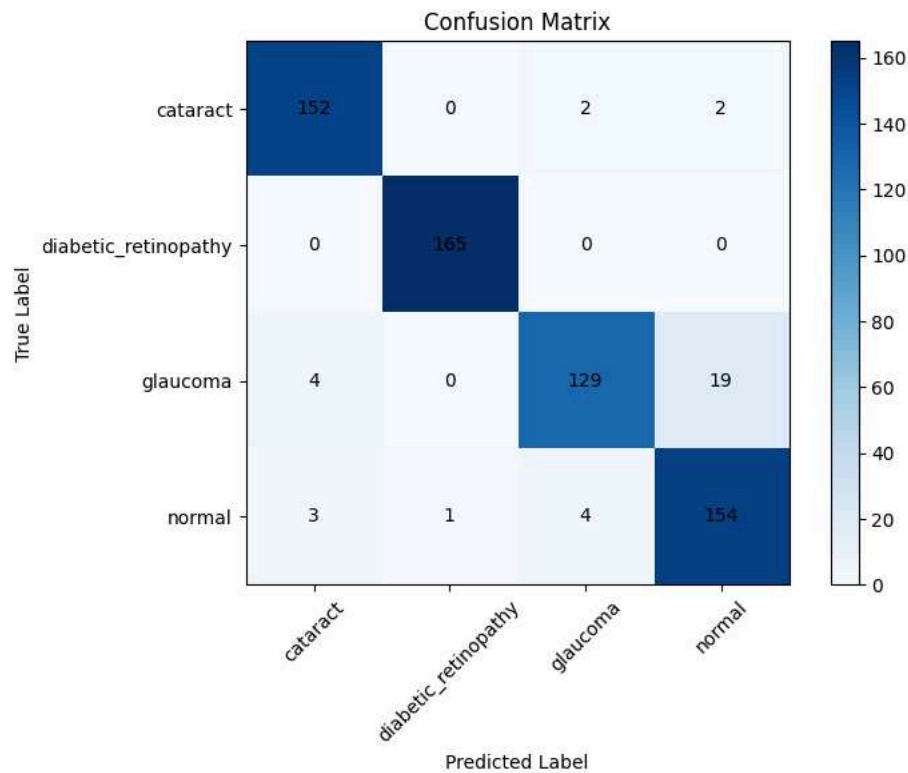
for i in range(len(classes)):
    for j in range(len(classes)):
        plt.text(
            j,
            i,
            cm[i, j],
            ha="center",
            va="center",
            color="black"
        )

plt.xlabel("Predicted Label")
plt.ylabel("True Label")

plt.tight_layout()

plt.show()

```



```
import numpy as np

model.eval()

images, labels = next(iter(test_loader))

images = images.to(device)

outputs = model(images)

_, predicted = torch.max(outputs, 1)

images = images.cpu()

plt.figure(figsize=(15,10))

for i in range(9):

    plt.subplot(3,3,i+1)

    img = images[i].permute(1,2,0).numpy()

    img = img * np.array([0.229,0.224,0.225]) + np.array([0.485,0.456,0.406])

    img = np.clip(img,0,1)

    plt.imshow(img)

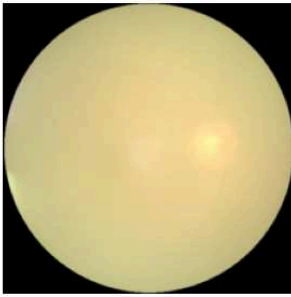
    plt.title(
        f"True: {classes[labels[i]]}\nPred: {classes[predicted[i]]}"
    )

    plt.axis("off")

plt.tight_layout()

plt.show()
```

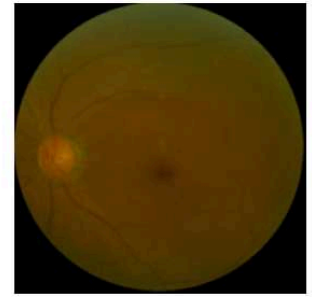
True: cataract
Pred: cataract



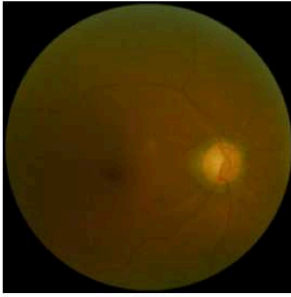
True: cataract
Pred: cataract



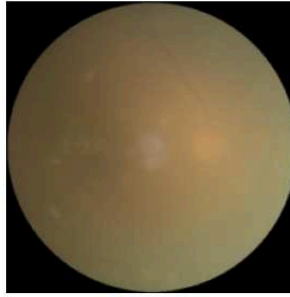
True: cataract
Pred: cataract



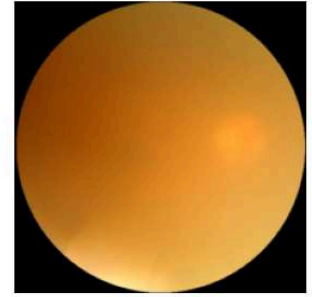
True: cataract
Pred: normal



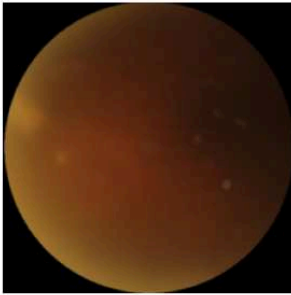
True: cataract
Pred: cataract



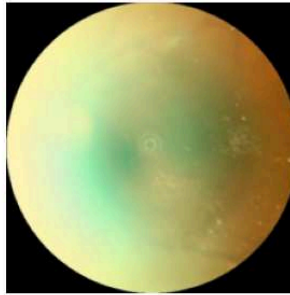
True: cataract
Pred: cataract



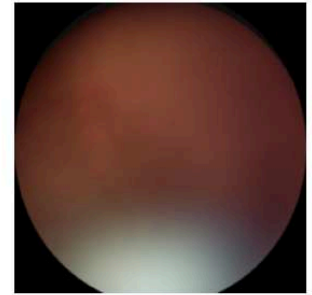
True: cataract
Pred: cataract



True: cataract
Pred: cataract



True: cataract
Pred: cataract



```
# =====  
# ResNet18 Model  
# =====  
  
from torchvision import models  
import torch.nn as nn  
  
num_classes = len(classes)  
  
resnet18 = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)  
  
resnet18.fc = nn.Linear(  
    resnet18.fc.in_features,  
    num_classes  
)  
  
resnet18 = resnet18.to(device)
```

Downloading: "<https://download.pytorch.org/models/resnet18-f37072fd.pth>" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd
100%|██████████| 44.7M/44.7M [00:00<00:00, 86.3MB/s]

```
criterion = nn.CrossEntropyLoss()  
  
optimizer = torch.optim.AdamW(  
    resnet18.parameters())
```

```

        resnet18.parameters(),
        lr=0.001
    )

    scheduler = torch.optim.lr_scheduler.StepLR(
        optimizer,
        step_size=3,
        gamma=0.97
    )

```

```

resnet18, train_losses_r, valid_losses_r, train_acc_r, valid_acc_r = train_model(
    resnet18,
    train_loader,
    validation_loader,
    criterion,
    optimizer,
    scheduler,
    num_epochs=30
)

```

```

Valid Loss : 0.2490
Train Accuracy : 0.9373
Valid Accuracy : 0.9084
-----

```

```

Epoch 17/30
Train Loss : 0.1771
Valid Loss : 1.0529
Train Accuracy : 0.9339
Valid Accuracy : 0.7551
-----

```

```

Epoch 18/30
Train Loss : 0.2046
Valid Loss : 0.2092
Train Accuracy : 0.9251
Valid Accuracy : 0.9258
-----

```

```

Epoch 19/30
Train Loss : 0.1568
Valid Loss : 0.2103
Train Accuracy : 0.9427
Valid Accuracy : 0.9258
-----

```

```

Epoch 20/30
Train Loss : 0.1658
Valid Loss : 0.2583
Train Accuracy : 0.9366
Valid Accuracy : 0.9052
-----

```

```

Epoch 21/30
Train Loss : 0.1493
Valid Loss : 0.1653
Train Accuracy : 0.9478
Valid Accuracy : 0.9463
-----

```

```

Epoch 22/30
Train Loss : 0.1649
Valid Loss : 0.1844
Train Accuracy : 0.9410
Valid Accuracy : 0.9368
-----

```

```

Epoch 23/30
Train Loss : 0.1409
Valid Loss : 0.1984
Train Accuracy : 0.9498
Valid Accuracy : 0.9258
-----

```

```

Epoch 24/30
Train Loss : 0.1354
Valid Loss : 0.2166
Train Accuracy : 0.9508
Valid Accuracy : 0.9305
-----

```

```

Epoch 25/30
Train Loss : 0.1364
Valid Loss : 0.2098
Train Accuracy : 0.9518
Valid Accuracy : 0.9305
-----

```

```

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def evaluate_model(model, test_loader):

    model.eval()

```

```

y_true = []
y_pred = []

with torch.no_grad():

    for images, labels in test_loader:

        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)

        _, predicted = torch.max(outputs, 1)

        y_true.extend(labels.cpu().numpy())
        y_pred.extend(predicted.cpu().numpy())

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

    return accuracy, precision, recall, f1

```

```

resnet_acc, resnet_pre, resnet_rec, resnet_f1 = evaluate_model(
    resnet18,
    test_loader
)

print("Accuracy :", resnet_acc)
print("Precision :", resnet_pre)
print("Recall :", resnet_rec)
print("F1 :", resnet_f1)

```

```

Accuracy : 0.9196850393700787
Precision : 0.9195770346920163
Recall : 0.9196850393700787
F1 : 0.9186002797900569

```

Start coding or generate with AI.

```

from sklearn.metrics import classification_report

print(classification_report(
    y_true,
    y_pred,
    target_names=classes
))

```

	precision	recall	f1-score	support
cataract	0.96	0.97	0.97	156
diabetic_retinopathy	0.99	1.00	1.00	165
glaucoma	0.96	0.85	0.90	152
normal	0.88	0.95	0.91	162
accuracy			0.94	635
macro avg	0.95	0.94	0.94	635
weighted avg	0.95	0.94	0.94	635

```

from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(8,6))

plt.imshow(cm, cmap="Blues")

plt.title("ResNet18 Confusion Matrix")

plt.colorbar()

plt.xticks(range(len(classes)), classes, rotation=45)
plt.yticks(range(len(classes)), classes)

for i in range(len(classes)):
    for j in range(len(classes)):
        plt.text(j, i, cm[i, j],

```

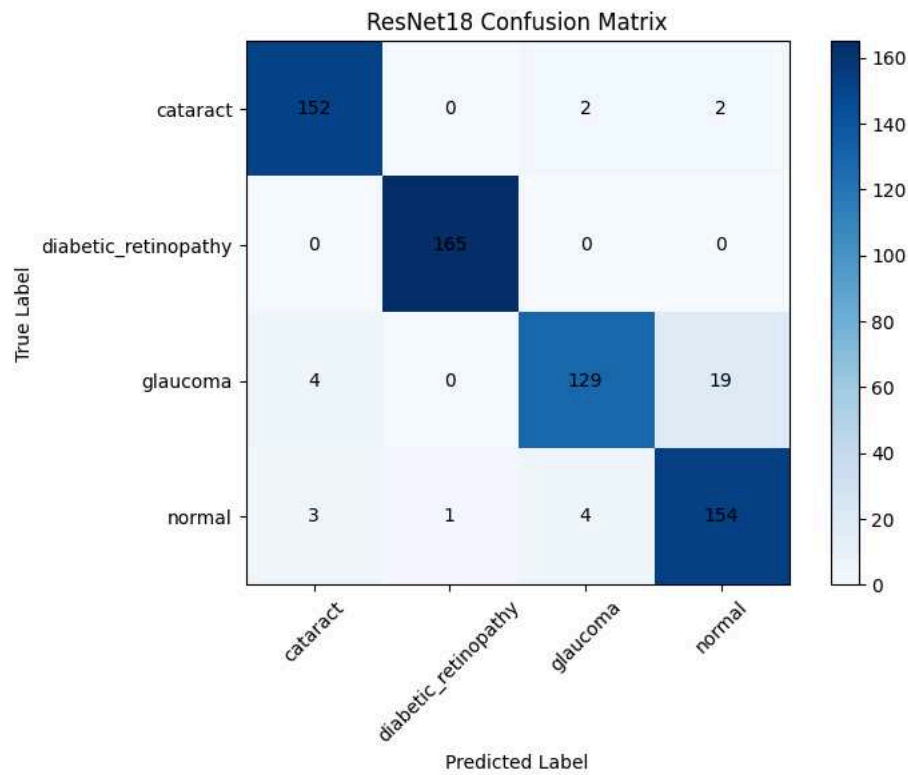
```

        ha="center",
        va="center",
        color="black")

plt.xlabel("Predicted Label")
plt.ylabel("True Label")

plt.tight_layout()
plt.show()

```



```

import matplotlib.pyplot as plt

plt.figure(figsize=(12,5))

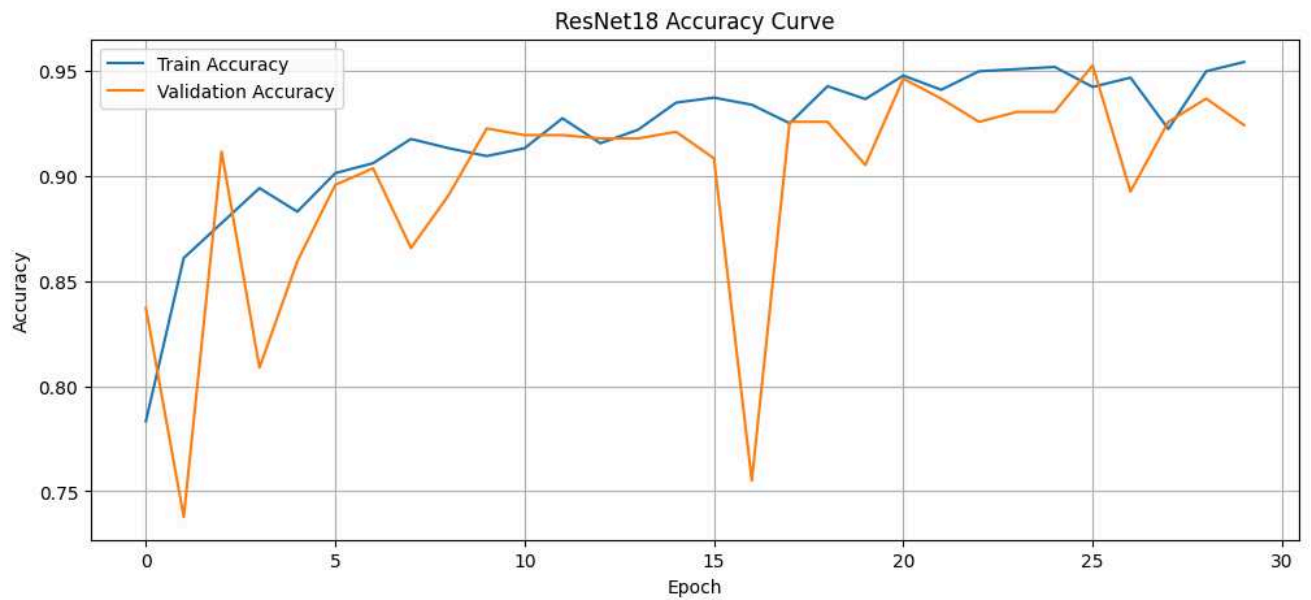
plt.plot(train_acc_r, label="Train Accuracy")
plt.plot(valid_acc_r, label="Validation Accuracy")

plt.title("ResNet18 Accuracy Curve")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")

plt.legend()
plt.grid(True)

plt.show()

```



```
import matplotlib.pyplot as plt

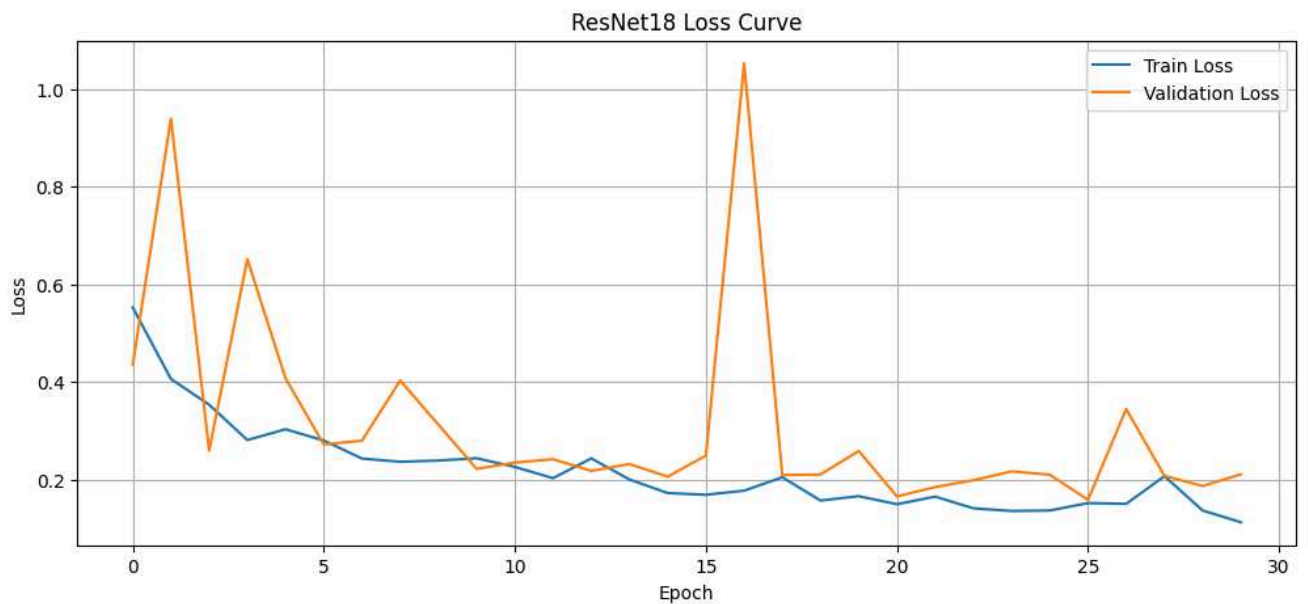
plt.figure(figsize=(12,5))

plt.plot(train_losses_r, label="Train Loss")
plt.plot(valid_losses_r, label="Validation Loss")

plt.title("ResNet18 Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")

plt.legend()
plt.grid(True)

plt.show()
```



```
# =====
# DenseNet121 Model
# =====

from torchvision import models
import torch.nn as nn

num_classes = len(classes)

densenet = models.densenet121(
```

```

weights=models.DenseNet121_Weights.DEFAULT
)

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

densenet = densenet.to(device)

print(densenet)

(relu1): ReLU(inplace=True)
(conv1): Conv2d(800, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
(norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(relu2): ReLU(inplace=True)
(conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
(denselayer11): _DenseLayer(
  (norm1): BatchNorm2d(832, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU(inplace=True)
  (conv1): Conv2d(832, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2): ReLU(inplace=True)
  (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
(denselayer12): _DenseLayer(
  (norm1): BatchNorm2d(864, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU(inplace=True)
  (conv1): Conv2d(864, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2): ReLU(inplace=True)
  (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
(denselayer13): _DenseLayer(
  (norm1): BatchNorm2d(896, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU(inplace=True)
  (conv1): Conv2d(896, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2): ReLU(inplace=True)
  (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
(denselayer14): _DenseLayer(
  (norm1): BatchNorm2d(928, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU(inplace=True)
  (conv1): Conv2d(928, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2): ReLU(inplace=True)
  (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
(denselayer15): _DenseLayer(
  (norm1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU(inplace=True)
  (conv1): Conv2d(960, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2): ReLU(inplace=True)
  (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
(denselayer16): _DenseLayer(
  (norm1): BatchNorm2d(992, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU(inplace=True)
  (conv1): Conv2d(992, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2): ReLU(inplace=True)
  (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
)
(norm5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
(classifier): Linear(in_features=1024, out_features=4, bias=True)

```

```

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.AdamW(
    densenet.parameters(),
    lr=0.001
)

scheduler = torch.optim.lr_scheduler.StepLR(
    optimizer,
    step_size=3,
    gamma=0.97
)

```

```
print(train_loader)
```

```
<torch.utils.data.dataloader.Dataloader object at 0x7d66af028c80>
```

Start coding or [generate](#) with AI.

```
from torch.utils.data import DataLoader
```

```
train_loader = DataLoader(  
    train_dataset,  
    batch_size=32,  
    shuffle=True  
)
```

```
validation_loader = DataLoader(  
    validation_dataset,  
    batch_size=32,  
    shuffle=False  
)
```

```
test_loader = DataLoader(  
    test_dataset,  
    batch_size=32,  
    shuffle=False  
)
```

```
train_loader = DataLoader(  
    train_dataset,  
    batch_size=32,  
    shuffle=True  
)
```

```
validation_loader = DataLoader(  
    validation_dataset,  
    batch_size=32,  
    shuffle=False  
)
```

```
test_loader = DataLoader(  
    test_dataset,  
    batch_size=32,  
    shuffle=False  
)
```

```
import torch
```

```
print(torch.cuda.is_available())  
print(torch.cuda.get_device_name(0))
```

```
x = torch.randn(1000, 1000).to(device)  
print(x.device)
```

```
True  
Tesla T4  
cuda:0
```

```
import torch
```

```
print("CUDA Available:", torch.cuda.is_available())
```

```
if torch.cuda.is_available():  
    print("GPU:", torch.cuda.get_device_name(0))
```

```
CUDA Available: True  
GPU: Tesla T4
```

```
densenet, train_losses_d, valid_losses_d, train_acc_d, valid_acc_d = train_model(  
    densenet,  
    train_loader,  
    validation_loader,  
    criterion,  
    optimizer,  
    scheduler,  
    num_epochs=30  
)
```



```
Epoch 1/30
Train Loss : 1.5007
Valid Loss : 1.4527
Train Accuracy : 0.2665
Valid Accuracy : 0.2543
-----
```

```
Epoch 2/30
Train Loss : 1.5012
Valid Loss : 1.4612
Train Accuracy : 0.2662
Valid Accuracy : 0.2686
-----
```

```
Epoch 3/30
Train Loss : 1.4990
Valid Loss : 1.4470
Train Accuracy : 0.2642
Valid Accuracy : 0.2575
-----
```

```
Epoch 4/30
Train Loss : 1.5046
Valid Loss : 1.4460
Train Accuracy : 0.2581
Valid Accuracy : 0.2749
-----
```

```
Epoch 5/30
Train Loss : 1.5045
Valid Loss : 1.4497
Train Accuracy : 0.2631
Valid Accuracy : 0.2670
-----
```

```
Epoch 6/30
Train Loss : 1.5082
Valid Loss : 1.4488
Train Accuracy : 0.2659
Valid Accuracy : 0.2622
-----
```

```
Epoch 7/30
Train Loss : 1.5027
Valid Loss : 1.4458
Train Accuracy : 0.2648
Valid Accuracy : 0.2780
-----
```

```
Epoch 8/30
Train Loss : 1.4991
Valid Loss : 1.4418
Train Accuracy : 0.2692
Valid Accuracy : 0.2686
-----
```

```
Epoch 9/30
Train Loss : 1.5018
Valid Loss : 1.4505
Train Accuracy : 0.2669
Valid Accuracy : 0.2622
-----
```

```
Epoch 10/30
Train Loss : 1.4993
Valid Loss : 1.4536
Train Accuracy : 0.2696
Valid Accuracy : 0.2654
-----
```

```
Epoch 11/30
Train Loss : 1.4992
Valid Loss : 1.4508
Train Accuracy : 0.2652
Valid Accuracy : 0.2765
-----
```

```
Epoch 12/30
Train Loss : 1.5018
Valid Loss : 1.4506
```

```
from torchvision import models
import torch.nn as nn

num_classes = len(classes)

densenet = models.densenet121(
    weights=models.DenseNet121_Weights.DEFAULT
)

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

densenet = densenet.to(device)
```

```
print("DenseNet121 Ready")
```

```
DenseNet121 Ready  
Valid Accuracy : 0.2559
```

```
criterion = nn.CrossEntropyLoss()
```

```
optimizer = torch.optim.AdamW(  
    densenet.parameters(),  
    lr=1e-4,  
    weight_decay=1e-4  
)
```

```
scheduler = torch.optim.lr_scheduler.StepLR(  
    optimizer,  
    step_size=5,  
    gamma=0.1  
)
```

```
train_loss : 1.5012
```

```
densenet, train_losses_d, valid_losses_d, train_acc_d, valid_acc_d = train_model(  
    densenet,  
    train_loader,  
    validation_loader,  
    criterion,  
    optimizer,  
    scheduler,  
    num_epochs=30  
)
```

```
Epoch 20/30  
Train Loss : 1.5056  
Valid Loss : 0.4816  
Train Accuracy : 0.2638  
Valid Accuracy : 0.8301
```

```
Epoch 21/30  
Train Loss : 1.5046  
Valid Loss : 0.4460  
Train Accuracy : 0.2686  
Valid Accuracy : 0.9479
```

```
Epoch 22/30  
Train Loss : 1.5032  
Valid Loss : 0.4543  
Train Accuracy : 0.3325  
Valid Accuracy : 0.9400
```

```
Epoch 23/30  
Train Loss : 0.1730  
Valid Loss : 0.1884  
Train Accuracy : 0.9430  
Valid Accuracy : 0.9368
```

```
Epoch 3/30  
Train Loss : 0.1561  
Valid Loss : 0.1865
```

```
Train Accuracy : 0.9434  
Valid Accuracy : 0.9368
```

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/densenet.py in forward(self, init_features)
```

```
Epoch 20/30  
new_features = layer(features)
```

```
Train Loss : 0.0979  
features.append(new_features)
```

```
Valid Loss : 0.1787  
return torch.cat(features, 1)
```

```
Train Accuracy : 0.9681
```

```
Valid Accuracy : 0.9384
```

```
KeyboardInterrupt:
```

```
Train Loss : 0.0885  
Valid Loss : 0.1875  
Train Accuracy : 0.9749  
Valid Accuracy : 0.9368
```

```
Epoch 8/30  
Train Loss : 0.0859  
Valid Loss : 0.1748  
Train Accuracy : 0.9749  
Valid Accuracy : 0.9431
```

```
Epoch 9/30  
Train Loss : 0.0801  
Valid Loss : 0.1782  
Train Accuracy : 0.9820  
Valid Accuracy : 0.9463
```

```
Epoch 10/30
Train Loss : 0.0711
Valid Loss : 0.1669
Train Accuracy : 0.0707
```

```
dense_acc, dense_pre, dense_rec, dense_f1 = evaluate_model(
    densenet,
    test_loader
)

print("Accuracy :", dense_acc)
print("Precision :", dense_pre)
print("Recall :", dense_rec)
print("F1 :", dense_f1)
```

```
-----
NameError                                Traceback (most recent call last)
/tmp/ipykernel_973/3264945341.py in <cell line: 0>()
----> 1 dense_acc, dense_pre, dense_rec, dense_f1 = evaluate_model(
      2     densenet,
      3     test_loader
      4 )
      5

NameError: name 'evaluate_model' is not defined
```

Next steps: [Explain error](#)

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def evaluate_model(model, test_loader):

    model.eval()

    y_true = []
    y_pred = []

    with torch.no_grad():

        for images, labels in test_loader:

            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)

            _, predicted = torch.max(outputs, 1)

            y_true.extend(labels.cpu().numpy())
            y_pred.extend(predicted.cpu().numpy())

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

    return accuracy, precision, recall, f1
```

```
dense_acc, dense_pre, dense_rec, dense_f1 = evaluate_model(
    densenet,
    test_loader
)

print("Accuracy :", dense_acc)
print("Precision :", dense_pre)
print("Recall :", dense_rec)
print("F1 :", dense_f1)
```

```
Accuracy : 0.937007874015748
Precision : 0.9368498589001123
Recall : 0.937007874015748
F1 : 0.9368699524571037
```

```
from sklearn.metrics import classification_report

print(classification_report(
    y_true,
    y_pred,
    target_names=classes
))
```

	precision	recall	f1-score	support
cataract	0.90	0.96	0.93	156
diabetic_retinopathy	0.99	1.00	1.00	165
glaucoma	0.94	0.64	0.77	152
normal	0.78	0.96	0.86	162
accuracy			0.89	635
macro avg	0.90	0.89	0.89	635
weighted avg	0.90	0.89	0.89	635

```

from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(8,6))
plt.imshow(cm, cmap="Blues")
plt.title("DenseNet121 Confusion Matrix")
plt.colorbar()

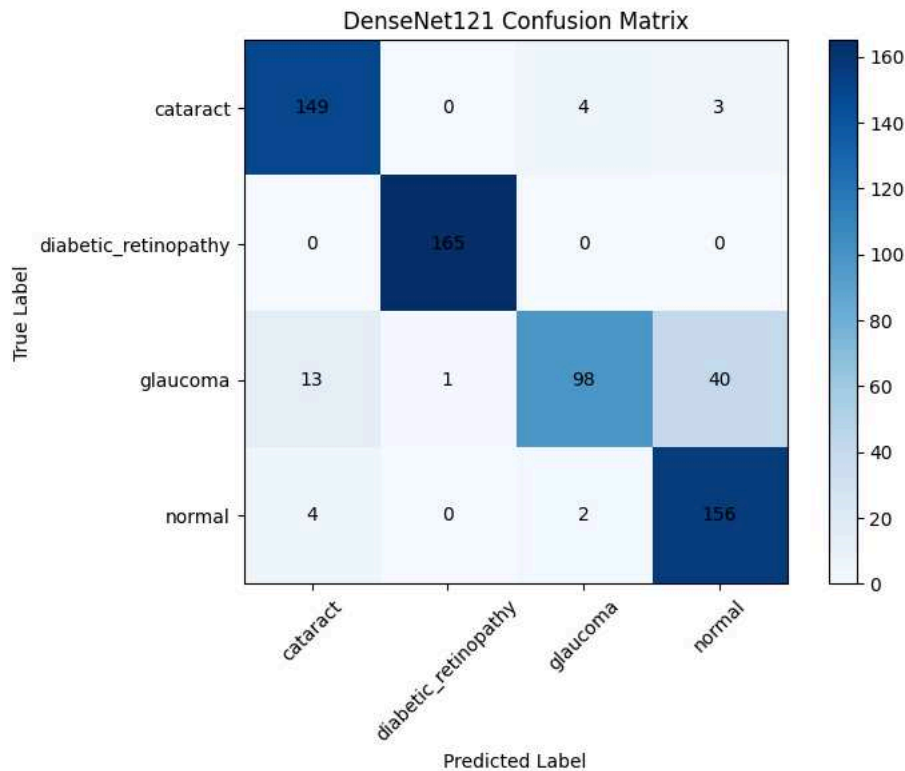
plt.xticks(range(len(classes)), classes, rotation=45)
plt.yticks(range(len(classes)), classes)

for i in range(len(classes)):
    for j in range(len(classes)):
        plt.text(j, i, cm[i, j],
                ha="center",
                va="center",
                color="black")

plt.xlabel("Predicted Label")
plt.ylabel("True Label")

plt.tight_layout()
plt.show()

```



```

import matplotlib.pyplot as plt

plt.figure(figsize=(12,5))

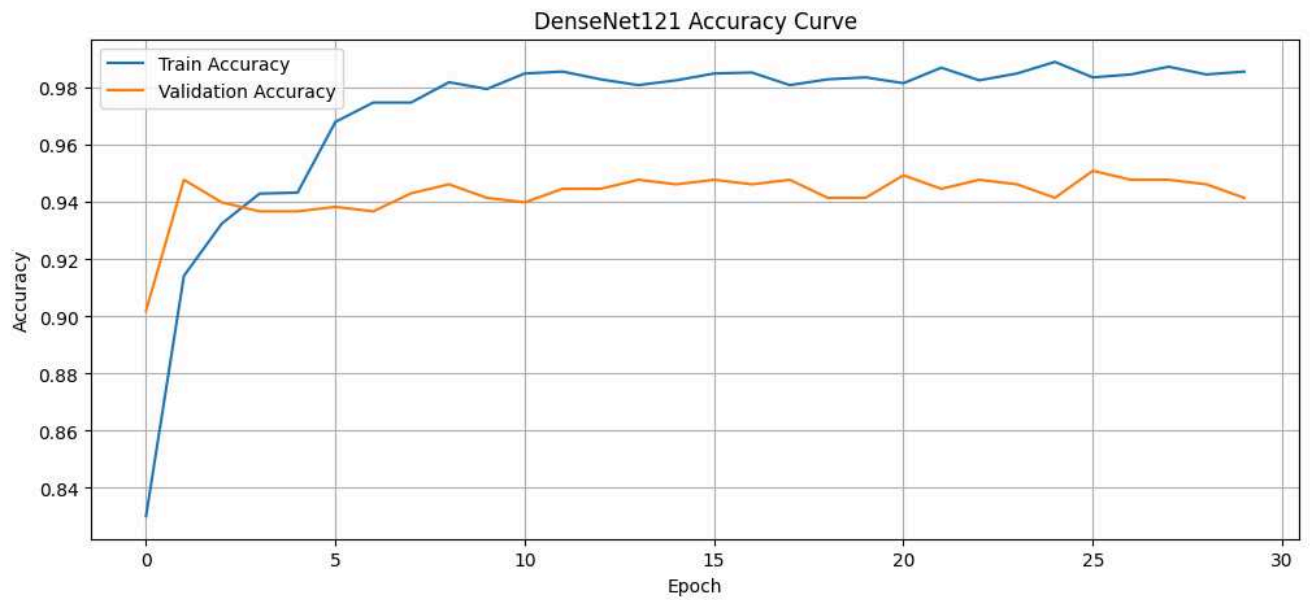
plt.plot(train_acc_d, label="Train Accuracy")
plt.plot(valid_acc_d, label="Validation Accuracy")

plt.title("DenseNet121 Accuracy Curve")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")

```

```
plt.legend()
plt.grid(True)

plt.show()
```



```
import matplotlib.pyplot as plt

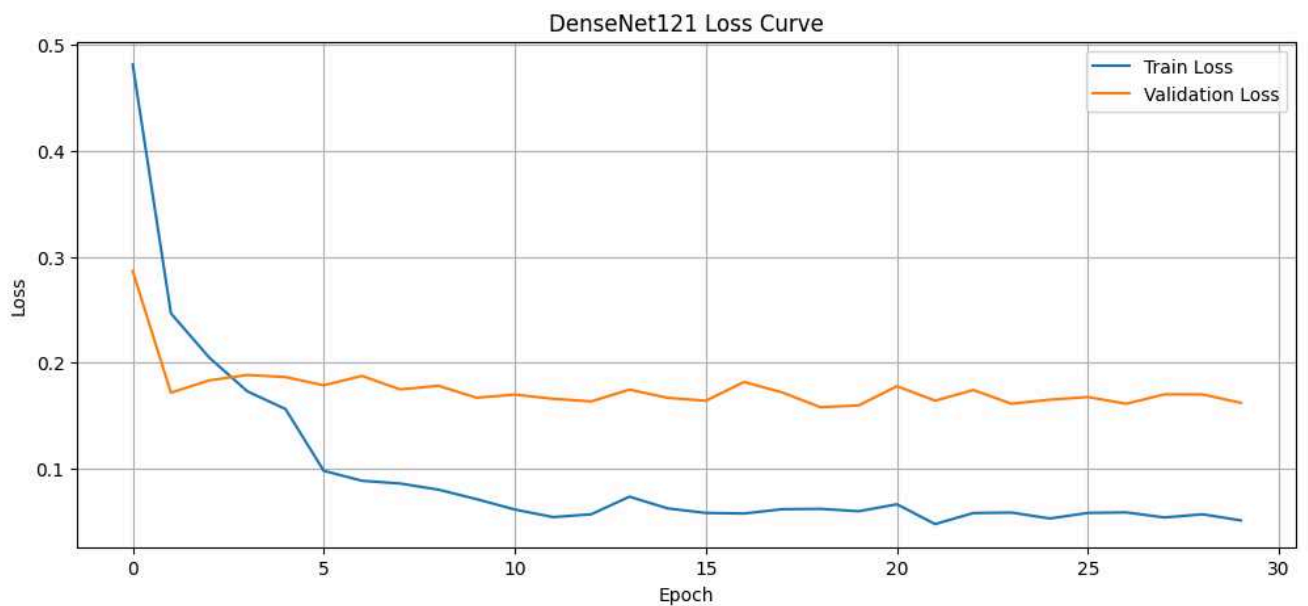
plt.figure(figsize=(12,5))

plt.plot(train_losses_d, label="Train Loss")
plt.plot(valid_losses_d, label="Validation Loss")

plt.title("DenseNet121 Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("Loss")

plt.legend()
plt.grid(True)

plt.show()
```



```
eff_acc, eff_pre, eff_rec, eff_f1 = evaluate_model(
    efficientnet,
    test_loader
)
```

```
print(eff_acc, eff_pre, eff_rec, eff_f1)
```

```
-----
NameError                                Traceback (most recent call last)
/tmp/ipykernel_973/3215516600.py in <cell line: 0>()
      1 eff_acc, eff_pre, eff_rec, eff_f1 = evaluate_model(
----> 2     efficientnet,
      3     test_loader
      4 )
      5

NameError: name 'efficientnet' is not defined
```

Next steps: [Explain error](#)

%whos

Variable	Type	Data/Info
Dataloader	type	<class 'torch.utils.data.dataloader.Dataloader'>
Image	module	<module 'PIL.Image' from <...>t-packages/PIL/Image.py'>
accuracy	float	0.8976377952755905
accuracy_score	function	<function accuracy_score at 0x7d66afa37e20>
class_path	str	/content/dataset/glaucoma
classes	list	n=4
classification_report	function	<function classification_<...>report at 0x7d66afaf9300>
cls	str	glaucoma
cm	ndarray	4x4: 16 elems, type 'int64', 128 bytes
confusion_matrix	function	<function confusion_matrix at 0x7d66afa37f60>
copy	module	<module 'copy' from '/usr<...>/lib/python3.12/copy.py'>
count	int	152
criterion	CrossEntropyLoss	CrossEntropyLoss()
dataset_path	str	/content
datasets	module	<module 'torchvision.data<...>on/datasets/_init_.py'>
dense_acc	float	0.937007874015748
dense_f1	float	0.9368699524571037
dense_pre	float	0.9368498589001123
dense_rec	float	0.937007874015748
densenet	DenseNet	DenseNet(\n (features): <...>features=4, bias=True)\n
device	device	cuda
dirs	list	n=0
evaluate_model	function	<function evaluate_model at 0x7d66a5fe36a0>
f1	float	0.8945562288021686
f1_score	function	<function f1_score at 0x7d66afaf8720>
files	list	n=6
i	int	3
images	Tensor	tensor([[[[-2.1179, -2.11<...>\n device='cuda:0')]
img	str	_239_5512637.jpg
j	int	3
labels	Tensor	tensor([3, 3, 3, 3, 3, 3,<...>, 3, 3], device='cuda:0')
model	EfficientNet	EfficientNet(\n (feature<...>res=4, bias=True)\n)\n
models	module	<module 'torchvision.mode<...>sion/models/_init_.py'>
nn	module	<module 'torch.nn' from '<...>es/torch/nn/_init_.py'>
np	module	<module 'numpy' from '/us<...>kages/numpy/_init_.py'>
num_classes	int	4
optim	module	<module 'torch.optim' fro<...>torch/optim/_init_.py'>
optimizer	AdamW	AdamW (\nParameter Group <...> weight_decay: 0.0001\n
os	module	<module 'os' (frozen)>
output_dir	str	/content/eye_dataset
outputs	Tensor	tensor([[-5.1389e-01, -1.<...>1e+00]], device='cuda:0')
pd	module	<module 'pandas' from '/u<...>ages/pandas/_init_.py'>
plt	module	<module 'matplotlib.pyplot<...>es/matplotlib/pyplot.py'>
precision	float	0.9029771412256224
precision_score	function	<function precision_score at 0x7d66afaf8f40>
predicted	Tensor	tensor([3, 3, 3, 3, 3, 3,<...>, 3, 3], device='cuda:0')
random	module	<module 'random' from '/u<...>ib/python3.12/random.py'>
recall	float	0.8976377952755905
recall_score	function	<function recall_score at 0x7d66afaf9080>
root	str	/content/sample_data
save_folder	str	/content/eye_dataset/test/glaucoma
scheduler	StepLR	<torch.optim.lr_scheduler<...>object at 0x7d66a5fab5f0>
seed	int	42
set_seed	function	<function set_seed at 0x7d66ba01aac0>
shutil	module	<module 'shutil' from '/u<...>ib/python3.12/shutil.py'>
sns	module	<module 'seaborn' from '<...>ges/seaborn/_init_.py'>

```
type(model)
```

```
torchvision.models.efficientnet.EfficientNet
def _wrapped_call_impl(*args, **kwargs)
```

As per the example above, an ``\_\_init\_\_()`` call to the parent class must be made before assignment on the child.

```
:ivar training: Boolean represents whether this module is in training or
                  evaluation mode.
:vartype training: bool
```

```
eff_acc, eff_pre, eff_rec, eff_f1 = evaluate_model(
    model,
    test_loader
)
```

```
print("Accuracy :", eff_acc)
print("Precision :", eff_pre)
print("Recall :", eff_rec)
print("F1 :", eff_f1)
```

```
Accuracy : 0.8976377952755905
Precision : 0.9029771412256224
Recall : 0.8976377952755905
F1 : 0.8945562288021686
```

```
print("resnet" in globals())
```

```
False
```

```
print(eff_acc, eff_pre, eff_rec, eff_f1)
```

```
0.8976377952755905 0.9029771412256224 0.8976377952755905 0.8945562288021686
```

```
eff_acc, eff_pre, eff_rec, eff_f1 = evaluate_model(
    model,
    test_loader
)
```

```
print(eff_acc, eff_pre, eff_rec, eff_f1)
```

```
0.8976377952755905 0.9029771412256224 0.8976377952755905 0.8945562288021686
```

```
import pandas as pd
```

```
results = pd.DataFrame({
    "Model": ["EfficientNet-B0", "ResNet18", "DenseNet121"],
    "Accuracy": [0.897637, 0.919685, 0.937008],
    "Precision": [0.902977, 0.919577, 0.936850],
    "Recall": [0.897637, 0.919685, 0.937008],
    "F1-Score": [0.894556, 0.918600, 0.936870]
})
```

```
results
```

	Model	Accuracy	Precision	Recall	F1-Score	
0	EfficientNet-B0	0.897637	0.902977	0.897637	0.894556	
1	ResNet18	0.919685	0.919577	0.919685	0.918600	
2	DenseNet121	0.937008	0.936850	0.937008	0.936870	

Next steps:

[Generate code with results](#)[New interactive sheet](#)

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(8,5))
```

```
plt.bar(results["Model"], results["Accuracy"])
```

```
plt.title("Model Accuracy Comparison")
plt.xlabel("Models")
plt.ylabel("Accuracy")
```

```
plt.ylim(0.85, 1.0)
```

```
for i, v in enumerate(results["Accuracy"]):
    plt.text(i, v + 0.003, f"{v:.3f}", ha='center')
```