```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.utils.data import DataLoader, random_split
from timm import create_model
import os
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, roc_auc_score
from PIL import Image
```

```
import torch
device = torch.device("cpu")
print("Using CPU for training. It will be slow but will work.")


# Paths
DATA_DIR = "/content/drive/MyDrive/CAPSTONE/Dataset_patches_small"  #
Folder containing 'IDC' and 'normal' subfolders
BATCH_SIZE = 32
EPOCHS = 10
LR = 0.001
IMG_SIZE = 256
NUM_CLASSES = 2  # IDC vs Normal
```

```
Using CPU for training. It will be slow but will work.
```

```
train_transform = transforms.Compose([
   transforms.Resize((IMG_SIZE, IMG_SIZE)),
   transforms.RandomHorizontalFlip(),
   transforms.RandomRotation(15),
   transforms.ColorJitter(brightness=0.2, contrast=0.2),
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
])

val_transform = transforms.Compose([
   transforms.Resize((IMG_SIZE, IMG_SIZE)),
   transforms.ToTensor(),
```

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

```
# Load full dataset
full_dataset = datasets.ImageFolder(root=DATA_DIR,
transform=train_transform)

# Split dataset (80% Train, 10% Val, 10% Test)
train_size = int(0.8 * len(full_dataset))
val_size = int(0.1 * len(full_dataset))
test_size = len(full_dataset) - train_size - val_size

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

# Create Data Loaders
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True, num_workers=4)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE,
shuffle=False, num_workers=4)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE,
shuffle=False, num_workers=4)

# Check dataset split
print(f"Train: {len(train_dataset)}, Val: {len(val_dataset)}, Test:
{len(test_dataset)}")
```

```
Train: 960, Val: 120, Test: 120
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624: UserWarning: This DataLoader will create 4 worker processes in total. Our sugges
  warnings.warn(
```

```
# Load EfficientNet-B0 with pretrained weights
model = create_model("efficientnet_b0", pretrained=True,
num_classes=NUM_CLASSES)
model.to(device)

# Define Loss & Optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=LR)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Col
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
model.safetensors: 100% 21.4M/21.4M [00:00<00:00, 49.1MB/s]
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from timm import create_model
import os

# Device setup
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using", device)

# Paths and params
DATA_DIR = "/content/drive/MyDrive/CAPSTONE/Dataset_patches_small"
BATCH_SIZE = 8
IMG_SIZE = 224
NUM_CLASSES = 2
EPOCHS = 15
LR = 0.0005

# Transforms
train_transform = transforms.Compose([
   transforms.Resize((IMG_SIZE, IMG_SIZE)),
   transforms.RandomHorizontalFlip(),
   transforms.RandomRotation(15),
   transforms.ColorJitter(0.2, 0.2),
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
])
val_transform = transforms.Compose([
   transforms.Resize((IMG_SIZE, IMG_SIZE)),
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
])

# Load dataset
full_dataset = datasets.ImageFolder(DATA_DIR,
transform=train_transform)
train_size = int(0.8 * len(full_dataset))
val_size = int(0.1 * len(full_dataset))
```

```
test_size = len(full_dataset) - train_size - val_size

train_ds, val_ds, test_ds = random_split(full_dataset, [train_size,
val_size, test_size])

train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE,
shuffle=True, num_workers=2)
val_loader = DataLoader(val_ds, batch_size=BATCH_SIZE, shuffle=False,
num_workers=2)
test_loader = DataLoader(test_ds, batch_size=BATCH_SIZE, shuffle=False,
num_workers=2)

# Load model
model = create_model("efficientnet_b0", pretrained=True,
num_classes=NUM_CLASSES)
model = model.to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=LR)

# ========================
# STAGE 2: TRAINING FUNCTION
# ========================
def train(model, train_loader, val_loader, optimizer, criterion,
epochs):
   best_acc = 0.0

   for epoch in range(epochs):
       model.train()
       running_loss, correct, total = 0.0, 0, 0

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

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

           running_loss += loss.item()
           preds = torch.argmax(outputs, dim=1)
```

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

        train_acc = correct / total
        print(f"Epoch [{epoch+1}/{epochs}] - Loss:
{running_loss/len(train_loader):.4f}, Train Acc: {train_acc:.4f}")

        # Validation
        model.eval()
        val_correct, val_total = 0, 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                preds = torch.argmax(outputs, dim=1)
                val_correct += (preds == labels).sum().item()
                val_total += labels.size(0)

        val_acc = val_correct / val_total
        print(f"Validation Acc: {val_acc:.4f}")

        if val_acc > best_acc:
            best_acc = val_acc
            torch.save(model.state_dict(),
"/content/drive/MyDrive/CAPSTONE/best_model.pth")
            print("Best model saved!")
train(model, train_loader, val_loader, optimizer, criterion, EPOCHS)
```

```
Epoch [1/15] - Loss: 1.9253, Train Acc: 0.7406
Validation Acc: 0.8417
Best model saved!
Epoch [2/15] - Loss: 1.0016, Train Acc: 0.8333
Validation Acc: 0.9167
Best model saved!
Epoch [3/15] - Loss: 0.9667, Train Acc: 0.8271
Validation Acc: 0.8583
Epoch [4/15] - Loss: 0.5746, Train Acc: 0.8604
Validation Acc: 0.8917
Epoch [5/15] - Loss: 0.4055, Train Acc: 0.8781
Validation Acc: 0.8667
Epoch [6/15] - Loss: 0.2747, Train Acc: 0.9031
Validation Acc: 0.9083
Epoch [7/15] - Loss: 0.2883, Train Acc: 0.9000
Validation Acc: 0.9333
Best model saved!
Epoch [8/15] - Loss: 0.2051, Train Acc: 0.9323
Validation Acc: 0.9250
Epoch [9/15] - Loss: 0.1522, Train Acc: 0.9458
Validation Acc: 0.9083
Epoch [10/15] - Loss: 0.1591, Train Acc: 0.9385
Validation Acc: 0.8917
Epoch [11/15] - Loss: 0.1405, Train Acc: 0.9563
Validation Acc: 0.9333
Epoch [12/15] - Loss: 0.1997, Train Acc: 0.9479
Validation Acc: 0.9250
Epoch [13/15] - Loss: 0.1725, Train Acc: 0.9417
Validation Acc: 0.9167
Epoch [14/15] - Loss: 0.1351, Train Acc: 0.9552
Validation Acc: 0.9250
Epoch [15/15] - Loss: 0.1710, Train Acc: 0.9458
Validation Acc: 0.8750
```

```python
import matplotlib.pyplot as plt

def train(model, train_loader, val_loader, optimizer, criterion,
epochs):
    best_acc = 0.0

    # Track metrics
    train_acc_list, val_acc_list = [], []
    train_loss_list, val_loss_list = [], []

    for epoch in range(epochs):
        model.train()
        running_loss, correct, total = 0.0, 0, 0

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

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

```
            running_loss += loss.item()
            preds = torch.argmax(outputs, dim=1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)

        train_acc = correct / total
        train_loss = running_loss / len(train_loader)
        train_acc_list.append(train_acc)
        train_loss_list.append(train_loss)

        # Validation
        model.eval()
        val_correct, val_total, val_loss = 0, 0, 0.0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                preds = torch.argmax(outputs, dim=1)

                val_loss += loss.item()
                val_correct += (preds == labels).sum().item()
                val_total += labels.size(0)

        val_acc = val_correct / val_total
        val_loss /= len(val_loader)

        val_acc_list.append(val_acc)
        val_loss_list.append(val_loss)

        print(f"Epoch [{epoch+1}/{epochs}] - Train Loss:
{train_loss:.4f}, Train Acc: {train_acc:.4f} | Val Loss:
{val_loss:.4f}, Val Acc: {val_acc:.4f}")

        if val_acc > best_acc:
            best_acc = val_acc
            torch.save(model.state_dict(),
"/content/drive/MyDrive/CAPSTONE/best_model.pth")
            print("✅ Best model saved!")

    # Save plots
```

```
    os.makedirs("/content/drive/MyDrive/CAPSTONE/metrics",
exist_ok=True)

    # Accuracy plot
    plt.figure(figsize=(10, 5))
    plt.plot(train_acc_list, label="Train Accuracy")
    plt.plot(val_acc_list, label="Validation Accuracy")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.title("Training vs Validation Accuracy")
    plt.legend()
    plt.grid()

plt.savefig("/content/drive/MyDrive/CAPSTONE/metrics/accuracy_curve.png
")
    plt.show()

    # Loss plot
    plt.figure(figsize=(10, 5))
    plt.plot(train_loss_list, label="Train Loss")
    plt.plot(val_loss_list, label="Validation Loss")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.title("Training vs Validation Loss")
    plt.legend()
    plt.grid()

plt.savefig("/content/drive/MyDrive/CAPSTONE/metrics/loss_curve.png")
    plt.show()



import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from timm import create_model
import os
import matplotlib.pyplot as plt


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

```
print("Using", device)


DATA_DIR = "/content/drive/MyDrive/CAPSTONE/Dataset_patches_small"
SAVE_DIR = "/content/drive/MyDrive/CAPSTONE"
METRIC_DIR = os.path.join(SAVE_DIR, "metrics")

BATCH_SIZE = 8
IMG_SIZE = 224
NUM_CLASSES = 2
EPOCHS = 30
LR = 0.0005


os.makedirs(METRIC_DIR, exist_ok=True)


train_transform = transforms.Compose([
   transforms.Resize((IMG_SIZE, IMG_SIZE)),
   transforms.RandomHorizontalFlip(),
   transforms.RandomRotation(15),
   transforms.ColorJitter(0.2, 0.2),
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
])
val_transform = transforms.Compose([
   transforms.Resize((IMG_SIZE, IMG_SIZE)),
   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=train_transform)
train_size = int(0.8 * len(full_dataset))
val_size = int(0.1 * len(full_dataset))
test_size = len(full_dataset) - train_size - val_size

train_ds, val_ds, test_ds = random_split(full_dataset, [train_size,
val_size, test_size])
```

```
train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE,
shuffle=True, num_workers=2)
val_loader = DataLoader(val_ds, batch_size=BATCH_SIZE, shuffle=False,
num_workers=2)
test_loader = DataLoader(test_ds, batch_size=BATCH_SIZE, shuffle=False,
num_workers=2)


model = create_model("efficientnet_b0", pretrained=True,
num_classes=NUM_CLASSES)
model = model.to(device)


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


train_accuracies = []
val_accuracies = []
train_losses = []
val_losses = []


def train(model, train_loader, val_loader, optimizer, criterion,
epochs):
   best_acc = 0.0
   for epoch in range(epochs):
       model.train()
       running_loss, correct, total = 0.0, 0, 0

       for images, labels in train_loader:
           images, labels = images.to(device), labels.to(device)
           optimizer.zero_grad()
           outputs = model(images)
           loss = criterion(outputs, labels)
           loss.backward()
           optimizer.step()

           running_loss += loss.item()
           preds = torch.argmax(outputs, dim=1)
           correct += (preds == labels).sum().item()
           total += labels.size(0)
```

```
        train_acc = correct / total
        train_loss = running_loss / len(train_loader)
        train_accuracies.append(train_acc)
        train_losses.append(train_loss)

        print(f"Epoch [{epoch+1}/{epochs}] - Loss: {train_loss:.4f},
Train Acc: {train_acc:.4f}")


        model.eval()
        val_correct, val_total = 0, 0
        val_running_loss = 0.0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                preds = torch.argmax(outputs, dim=1)
                val_correct += (preds == labels).sum().item()
                val_running_loss += loss.item()

        val_acc = val_correct / len(val_loader.dataset)
        val_loss = val_running_loss / len(val_loader)
        val_accuracies.append(val_acc)
        val_losses.append(val_loss)

        print(f"Validation Acc: {val_acc:.4f}")

        if val_acc > best_acc:
            best_acc = val_acc
            torch.save(model.state_dict(), os.path.join(SAVE_DIR,
"best_model.pth"))
            print("✅ Best model saved!")


    plt.figure(figsize=(10, 5))
    plt.plot(train_accuracies, label='Train Accuracy')
    plt.plot(val_accuracies, label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Accuracy Curve')
    plt.legend()
    plt.grid(True)
```

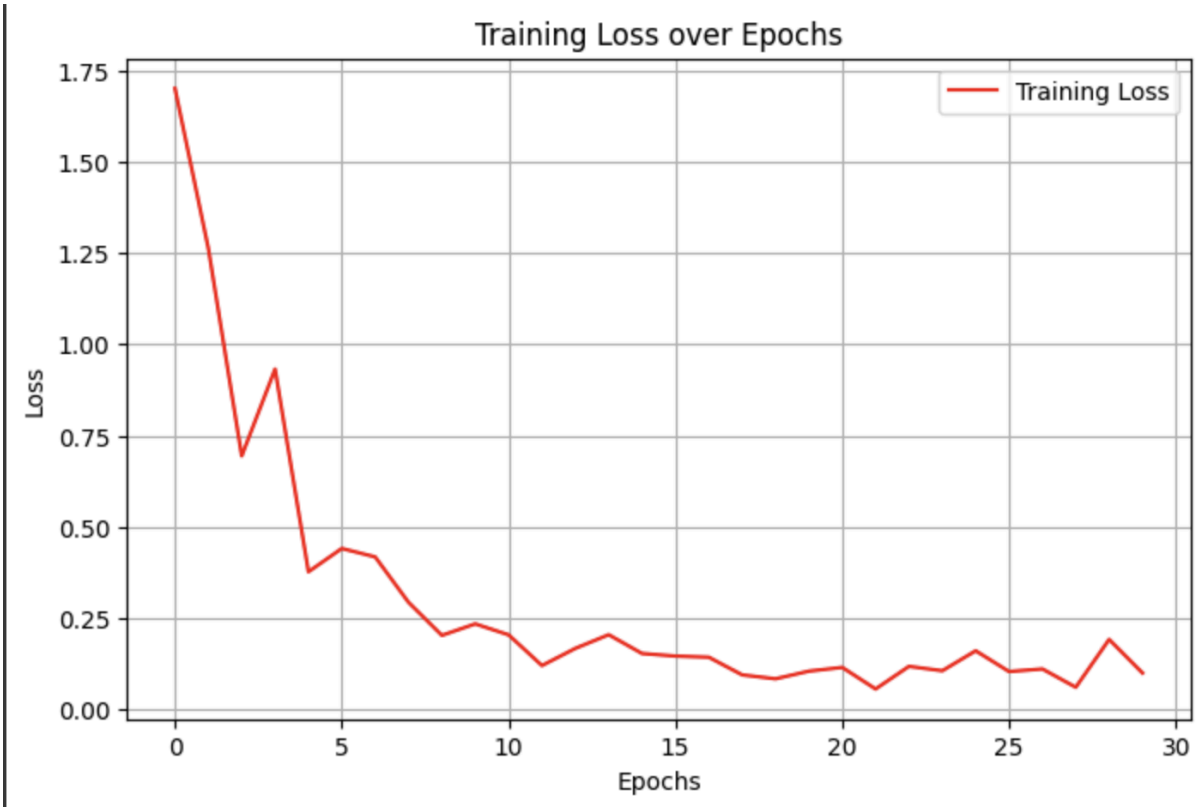
```
    plt.savefig(os.path.join(METRIC_DIR, "accuracy_curve.png"))
    plt.close()



    plt.figure(figsize=(10, 5))
    plt.plot(train_losses, label='Train Loss')
    plt.plot(val_losses, label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Loss Curve')
    plt.legend()
    plt.grid(True)
    plt.savefig(os.path.join(METRIC_DIR, "loss_curve.png"))
    plt.close()


train(model, train_loader, val_loader, optimizer, criterion, EPOCHS)
```

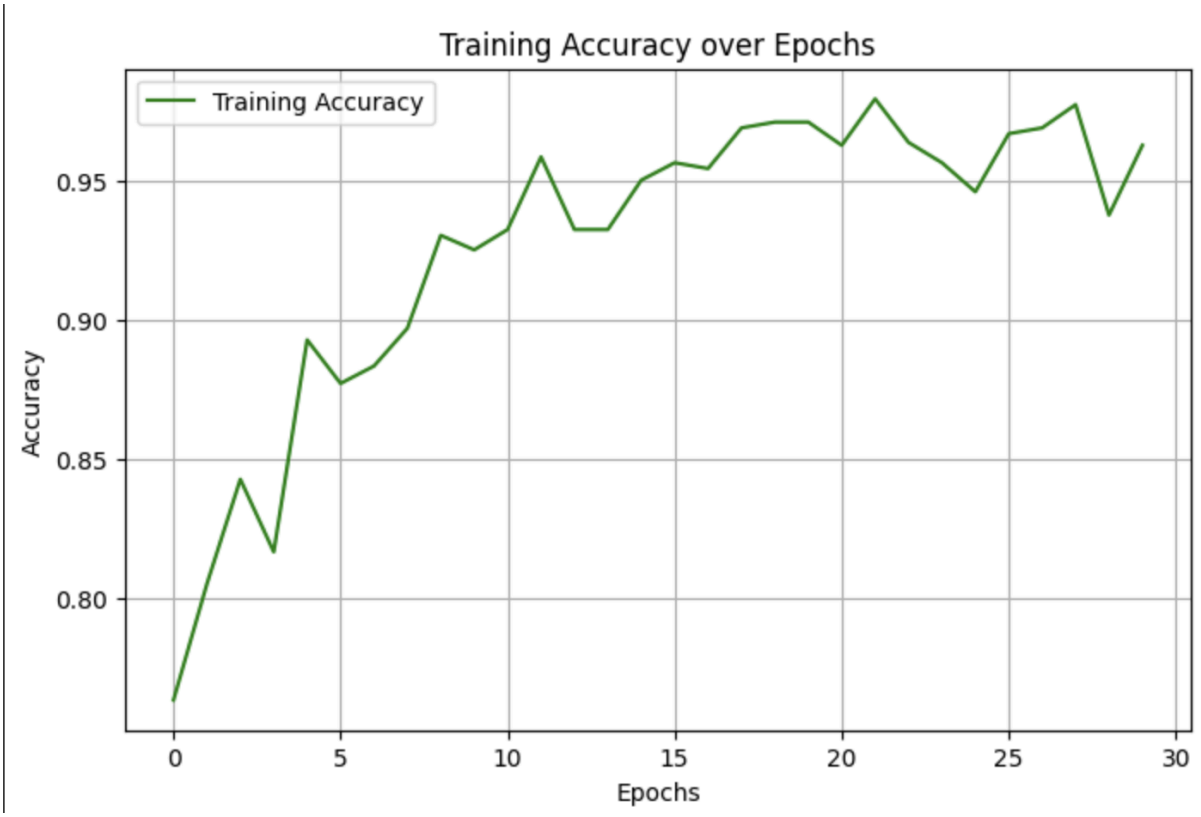
```
model.safetensors: 100% 21.4M/21.4M [00:00<00:00, 80.2MB/s]
Epoch [1/30] - Loss: 1.7011, Train Acc: 0.7635
Validation Acc: 0.7500
✅ Best model saved!
Epoch [2/30] - Loss: 1.2627, Train Acc: 0.8052
Validation Acc: 0.8583
✅ Best model saved!
Epoch [3/30] - Loss: 0.6943, Train Acc: 0.8427
Validation Acc: 0.7833
Epoch [4/30] - Loss: 0.9322, Train Acc: 0.8167
Validation Acc: 0.9000
✅ Best model saved!
Epoch [5/30] - Loss: 0.3769, Train Acc: 0.8927
Validation Acc: 0.9250
✅ Best model saved!
Epoch [6/30] - Loss: 0.4407, Train Acc: 0.8771
Validation Acc: 0.8750
Epoch [7/30] - Loss: 0.4176, Train Acc: 0.8833
Validation Acc: 0.8833
Epoch [8/30] - Loss: 0.2931, Train Acc: 0.8969
Validation Acc: 0.9250
Epoch [9/30] - Loss: 0.2026, Train Acc: 0.9302
Validation Acc: 0.9250
Epoch [10/30] - Loss: 0.2341, Train Acc: 0.9250
Validation Acc: 0.8833
Epoch [11/30] - Loss: 0.2040, Train Acc: 0.9323
Validation Acc: 0.9333
✅ Best model saved!
Epoch [12/30] - Loss: 0.1199, Train Acc: 0.9583
Validation Acc: 0.9333
Epoch [13/30] - Loss: 0.1670, Train Acc: 0.9323
Validation Acc: 0.9250
Epoch [14/30] - Loss: 0.2044, Train Acc: 0.9323
Validation Acc: 0.9167
```

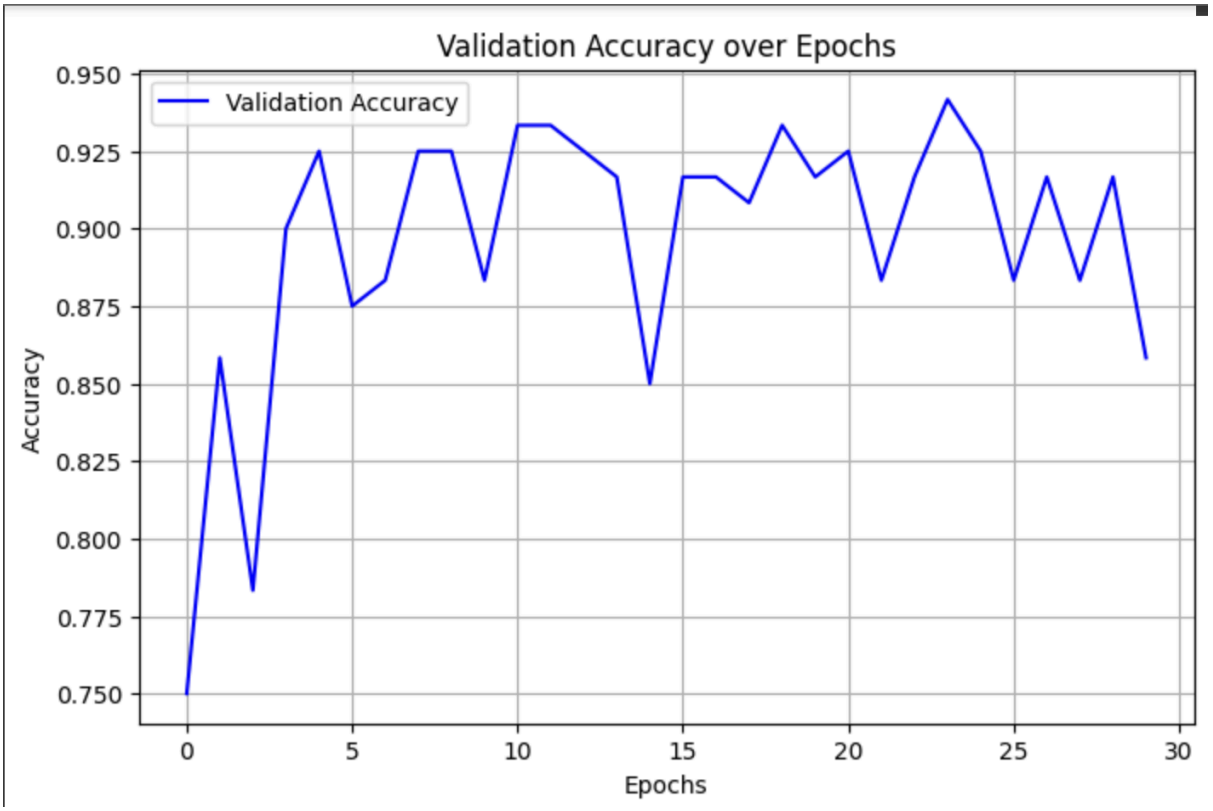
```
    Validation Acc: 0.9167
    Epoch [15/30] - Loss: 0.1529, Train Acc: 0.9500
    Validation Acc: 0.8500
    Epoch [16/30] - Loss: 0.1459, Train Acc: 0.9563
    Validation Acc: 0.9167
    Epoch [17/30] - Loss: 0.1428, Train Acc: 0.9542
    Validation Acc: 0.9167
    Epoch [18/30] - Loss: 0.0944, Train Acc: 0.9688
    Validation Acc: 0.9083
    Epoch [19/30] - Loss: 0.0836, Train Acc: 0.9708
    Validation Acc: 0.9333
    Epoch [20/30] - Loss: 0.1046, Train Acc: 0.9708
    Validation Acc: 0.9167
    Epoch [21/30] - Loss: 0.1149, Train Acc: 0.9625
    Validation Acc: 0.9250
    Epoch [22/30] - Loss: 0.0557, Train Acc: 0.9792
    Validation Acc: 0.8833
    Epoch [23/30] - Loss: 0.1177, Train Acc: 0.9635
    Validation Acc: 0.9167
    Epoch [24/30] - Loss: 0.1057, Train Acc: 0.9563
    Validation Acc: 0.9417
    ✅ Best model saved!
    Epoch [25/30] - Loss: 0.1604, Train Acc: 0.9458
    Validation Acc: 0.9250
    Epoch [26/30] - Loss: 0.1035, Train Acc: 0.9667
    Validation Acc: 0.8833
    Epoch [27/30] - Loss: 0.1106, Train Acc: 0.9688
    Validation Acc: 0.9167
    Epoch [28/30] - Loss: 0.0604, Train Acc: 0.9771
    Validation Acc: 0.8833
    Epoch [29/30] - Loss: 0.1915, Train Acc: 0.9375
    Validation Acc: 0.9167
    Epoch [30/30] - Loss: 0.0996, Train Acc: 0.9625
    Validation Acc: 0.8583
```

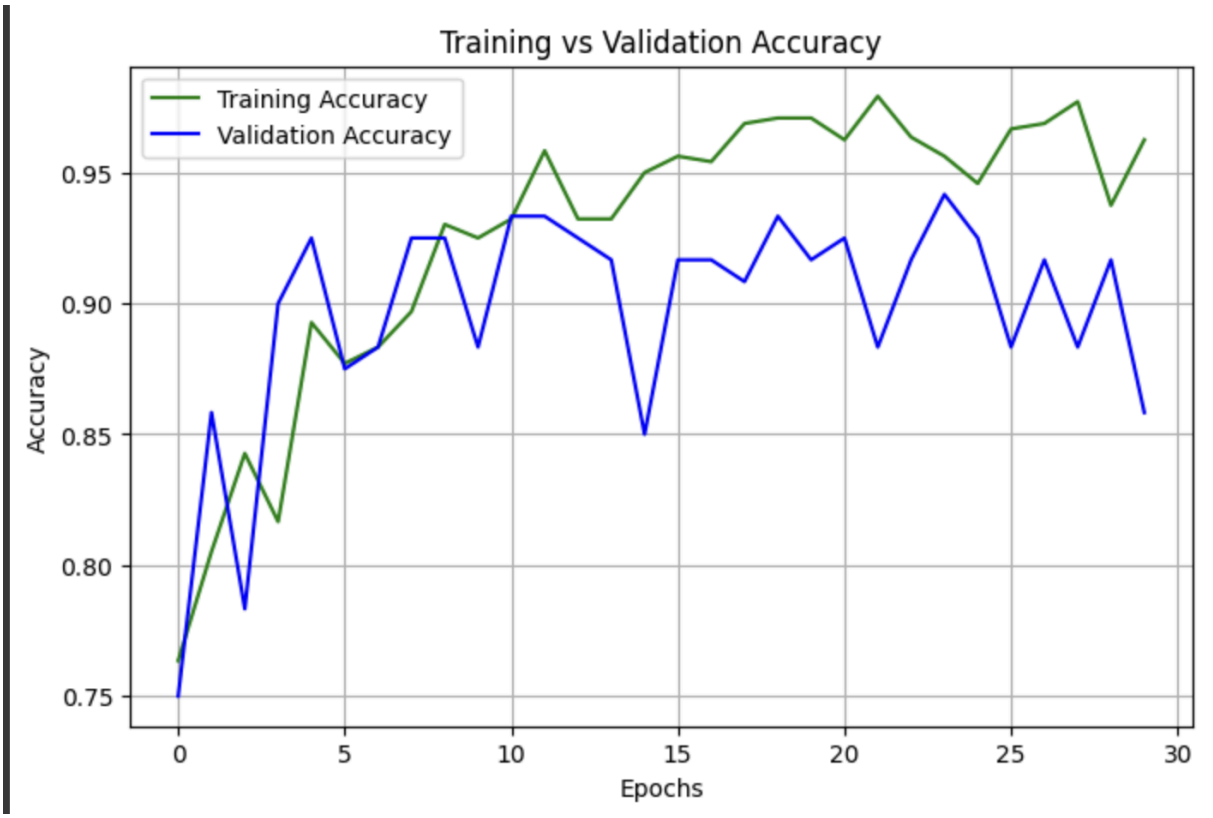
```python
import matplotlib.pyplot as plt
import os


metrics_dir = "/content/drive/MyDrive/CAPSTONE/metrics"
os.makedirs(metrics_dir, exist_ok=True)


plt.figure(figsize=(8, 5))
plt.plot(train_losses, label='Training Loss', color='red')
plt.title("Training Loss over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(metrics_dir, "training_loss.png"))
plt.show()


plt.figure(figsize=(8, 5))
plt.plot(train_accuracies, label='Training Accuracy', color='green')
plt.title("Training Accuracy over Epochs")
```

```
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(metrics_dir, "training_accuracy.png"))
plt.show()


plt.figure(figsize=(8, 5))
plt.plot(val_accuracies, label='Validation Accuracy', color='blue')
plt.title("Validation Accuracy over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(metrics_dir, "validation_accuracy.png"))
plt.show()


plt.figure(figsize=(8, 5))
plt.plot(train_accuracies, label='Training Accuracy', color='green')
plt.plot(val_accuracies, label='Validation Accuracy', color='blue')
plt.title("Training vs Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(metrics_dir,
"train_val_accuracy_comparison.png"))
plt.show()
```


Training Loss over Epochs
Training Loss
Loss
Epochs
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
25
30


Training Accuracy over Epochs
Training Accuracy
Accuracy
Epochs
0.95
0.90
0.85
0.80
0
5
10
15
20
25
30

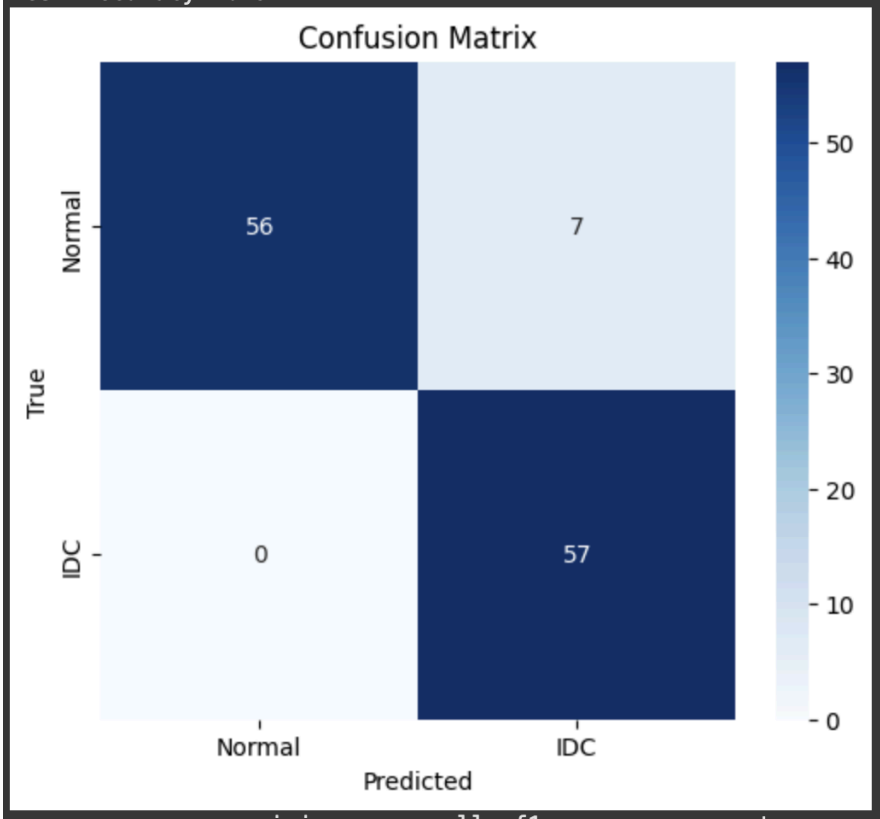

Validation Accuracy over Epochs
Validation Accuracy
Accuracy
Epochs
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0
5
10
15
20
25
30


Training vs Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.95
0.90
0.85
0.80
0.75
0
5
10
15
20
25
30

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score
import numpy as np
import os


os.makedirs("/content/drive/MyDrive/CAPSTONE/metrics", exist_ok=True)


model.load_state_dict(torch.load("/content/drive/MyDrive/CAPSTONE/best_
model.pth"))
model.eval()

all_preds = []
all_labels = []

with torch.no_grad():
   for images, labels in test_loader:
       images, labels = images.to(device), labels.to(device)
       outputs = model(images)
       preds = torch.argmax(outputs, dim=1)
       all_preds.extend(preds.cpu().numpy())
       all_labels.extend(labels.cpu().numpy())


acc = accuracy_score(all_labels, all_preds)
print(f"Test Accuracy: {acc:.4f}")


cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Normal', 'IDC'], yticklabels=['Normal', 'IDC'])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.savefig("/content/drive/MyDrive/CAPSTONE/metrics/confusion_matrix.p
ng")
plt.show()
```

```
report = classification_report(all_labels, all_preds,
target_names=['Normal', 'IDC'])
print(report)

with
open("/content/drive/MyDrive/CAPSTONE/metrics/classification_report.txt
", "w") as f:
   f.write(report)
```

```
Test Accuracy: 0.9417
```

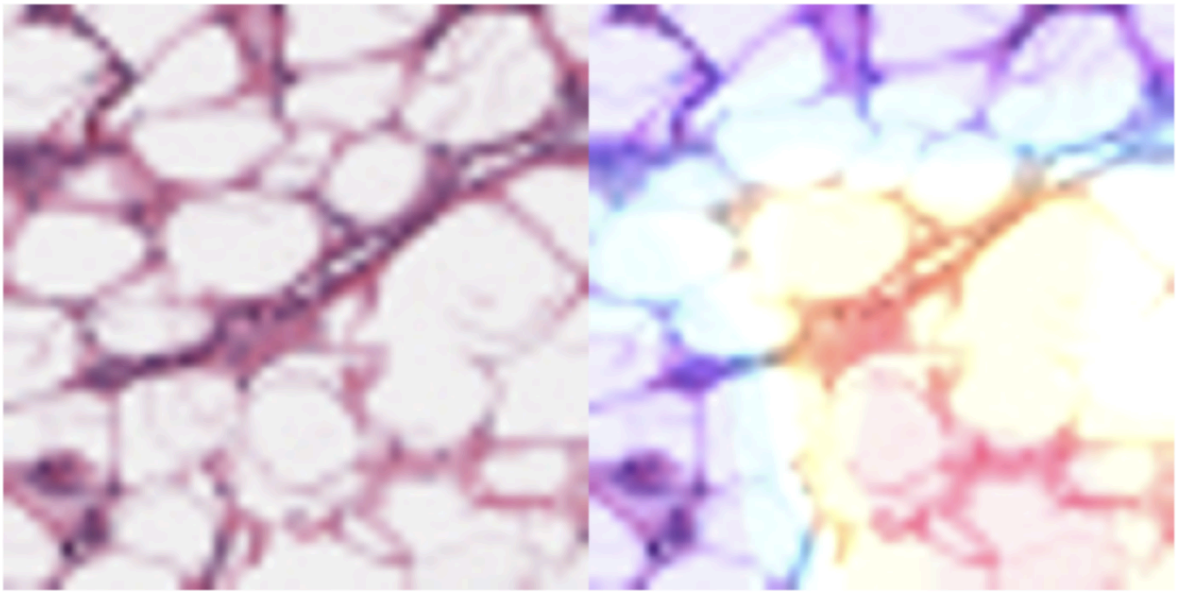



```
              precision    recall  f1-score   support

      Normal       1.00      0.89      0.94        63
         IDC       0.89      1.00      0.94        57

    accuracy                           0.94       120
   macro avg       0.95      0.94      0.94       120
weighted avg       0.95      0.94      0.94       120
```

```
import matplotlib.pyplot as plt
import cv2
import os


gradcam_dir = "/content/drive/MyDrive/CAPSTONE/gradcam"
```

```
gradcam_images = [f for f in os.listdir(gradcam_dir) if
f.endswith('.png')]


for img_name in gradcam_images:
   img_path = os.path.join(gradcam_dir, img_name)


   img = cv2.imread(img_path)
   img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)


   plt.figure(figsize=(8, 6))
   plt.imshow(img_rgb)
   plt.axis('off')
   plt.title(f"Grad-CAM for {img_name}")
   plt.show()
```

Grad-CAM for efficientnet_gradcam_8863_idx5_x1751_y2201_class0.png



Grad-CAM for efficientnet_gradcam_8863_idx5_x1501_y601_class0.png