untitled18

October 17, 2024

[3]:
```
import torch
import torch.optim as optim
import torch.nn as nn
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, random_split
import timm
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, confusion_matrix, matthews_corrcoef
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
```

[4]:
```
######DEFINE THE PATH
data_dir = "/Users/krutikadeshmukh/Downloads/Oral Images Dataset 2/
↪original_data"
```

[5]:
```
#####DATA AUGMENTATION AND PREPROCESSING OF IMAGES FOR TRAINING
train_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(20),
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.
↪2),
    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
])

######ONLY BASIC PREPROCESSING AND TRANSFORMATION FOR TESTING
test_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
])
```

[6]:
```
######LOADING THE ORIGINAL DATASET
original_dataset = datasets.ImageFolder(data_dir, transform=train_transforms)
```

[7]:
```
####SPLIT THE DATASET INTO TRAINING (80%) AND TESTING (20%)
train_size = int(0.8 * len(original_dataset))
test_size = len(original_dataset) - train_size
train_dataset, test_dataset = random_split(original_dataset, [train_size,␣
↪test_size])
```

[8]:
```
####APPLYING DIFFERENT TRANSFORMS TO THE TEST DATASET
test_dataset.dataset.transform = test_transforms
```

[9]:
```
#####CREATING THE DATA LOADERS FOR TESTING AND TRAINING DATASET
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

[10]:
```
#####DEFINING THE MODEL
model = timm.create_model('efficientvit_b0', pretrained=False, num_classes=2)
```

[11]:
```
####PATH TO THE PTH FILE
pth_file_path = '/Users/krutikadeshmukh/Downloads/
↪efficientvit_b0_oral_disease_classifier.pth'
```

[12]:
```
####LOADING THE MODEL WEIGHTS
model_weights = torch.load(pth_file_path, map_location=torch.device('cpu'))
```

```
/var/folders/qm/w3fd9xt10b90v_xr6kfn5kkh0000gn/T/ipykernel_4973/2267464208.py:2:
FutureWarning: You are using `torch.load` with `weights_only=False` (the current
default value), which uses the default pickle module implicitly. It is possible
to construct malicious pickle data which will execute arbitrary code during
unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for
more details). In a future release, the default value for `weights_only` will be
flipped to `True`. This limits the functions that could be executed during
unpickling. Arbitrary objects will no longer be allowed to be loaded via this
mode unless they are explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control of the
loaded file. Please open an issue on GitHub for any issues related to this
experimental feature.
  model_weights = torch.load(pth_file_path, map_location=torch.device('cpu'))
```

[13]:
```
#####FILTERING OUT THE CLASSIFIER WEIGHTS
pretrained_dict = {k: v for k, v in model_weights.items() if "classifier" not␣
↪in k}
model.load_state_dict(pretrained_dict, strict=False)
```

[13]:
```
_IncompatibleKeys(missing_keys=['head.classifier.0.weight',
'head.classifier.1.weight', 'head.classifier.1.bias',
'head.classifier.4.weight', 'head.classifier.4.bias'], unexpected_keys=[])
```

[14]:
```
######DEFINE THE LOSS FUNCTION AND OPTIMIZER
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)
```

[15]:
```
####### FUNCTION TO TRAIN THE MODEL AND TRACK METRICS
def train_model(model, criterion, optimizer, train_loader, test_loader,␣
 ↪num_epochs=5):
    train_losses, test_losses = [], []
    train_accuracies, test_accuracies = [], []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct_train = 0
        total_train = 0

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

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

            # Calculate accuracy for training
            _, predicted = torch.max(outputs, 1)
            correct_train += (predicted == labels).sum().item()
            total_train += labels.size(0)

        train_loss = running_loss / len(train_loader)
        train_acc = correct_train / total_train
        train_losses.append(train_loss)
        train_accuracies.append(train_acc)

        # Validation phase
        model.eval()
        running_loss_test = 0.0
        correct_test = 0
        total_test = 0

        with torch.no_grad():
            for inputs, labels in test_loader:
```

```
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                running_loss_test += loss.item()

                # Calculate accuracy for validation
                _, predicted = torch.max(outputs, 1)
                correct_test += (predicted == labels).sum().item()
                total_test += labels.size(0)

        test_loss = running_loss_test / len(test_loader)
        test_acc = correct_test / total_test
        test_losses.append(test_loss)
        test_accuracies.append(test_acc)

        print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f},␣
↪Train Accuracy: {train_acc:.4f}, Test Loss: {test_loss:.4f}, Test Accuracy:␣
↪{test_acc:.4f}")

    # Plot training and validation loss
    plt.figure(figsize=(10, 5))
    plt.plot(train_losses, label='Train Loss')
    plt.plot(test_losses, label='Test Loss')
    plt.title('Loss over Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

    # Plot training and validation accuracy
    plt.figure(figsize=(10, 5))
    plt.plot(train_accuracies, label='Train Accuracy')
    plt.plot(test_accuracies, label='Test Accuracy')
    plt.title('Accuracy over Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```

[25]:
```
### FUNCTION TO EVALUATE THE MODEL
def evaluate_model(model, test_loader):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for inputs, labels in test_loader:
```

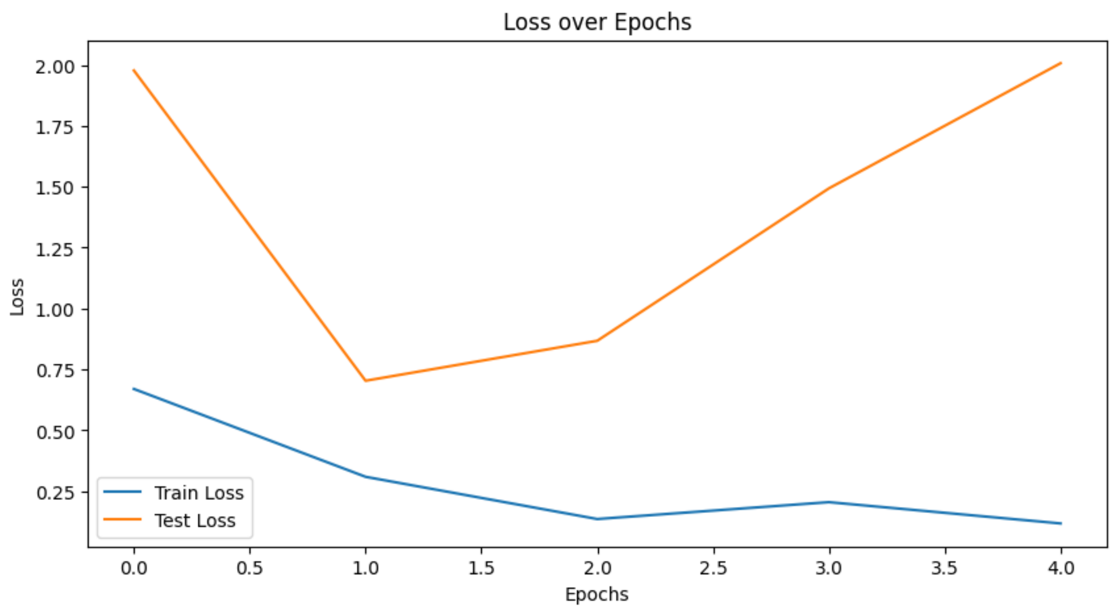
```
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    #### METRICS CALCULATION
    accuracy = accuracy_score(all_labels, all_preds)
    precision = precision_score(all_labels, all_preds, average='binary')
    recall = recall_score(all_labels, all_preds, average='binary')
    f1 = f1_score(all_labels, all_preds, average='binary')
    mcc = matthews_corrcoef(all_labels, all_preds)

    print(f"Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall:␣
 ↪{recall:.4f}, F1 Score: {f1:.4f}, MCC: {mcc:.4f}")

    ##### PLOTTING THE CONFUSION MATRIX
    cm = confusion_matrix(all_labels, all_preds)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Benign',␣
 ↪'Malignant'], yticklabels=['Benign', 'Malignant'])
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.title('Confusion Matrix')
    plt.show()
```

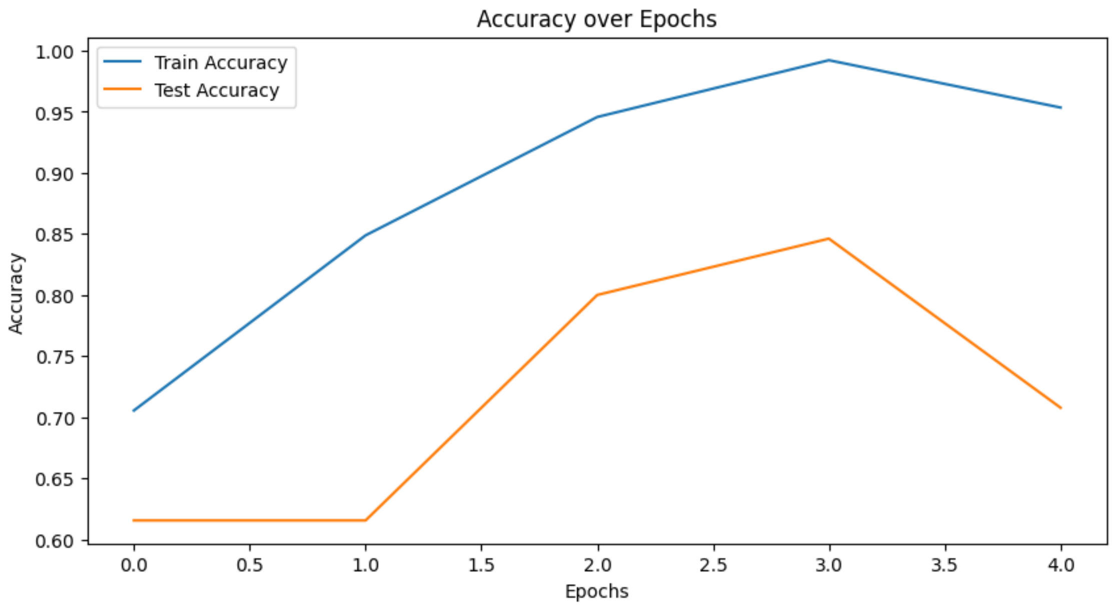
[26]:

```
##### STEP TO RUN TRAINING AND EVALUATION
train_model(model, criterion, optimizer, train_loader, test_loader,␣
 ↪num_epochs=5)
evaluate_model(model, test_loader)
```

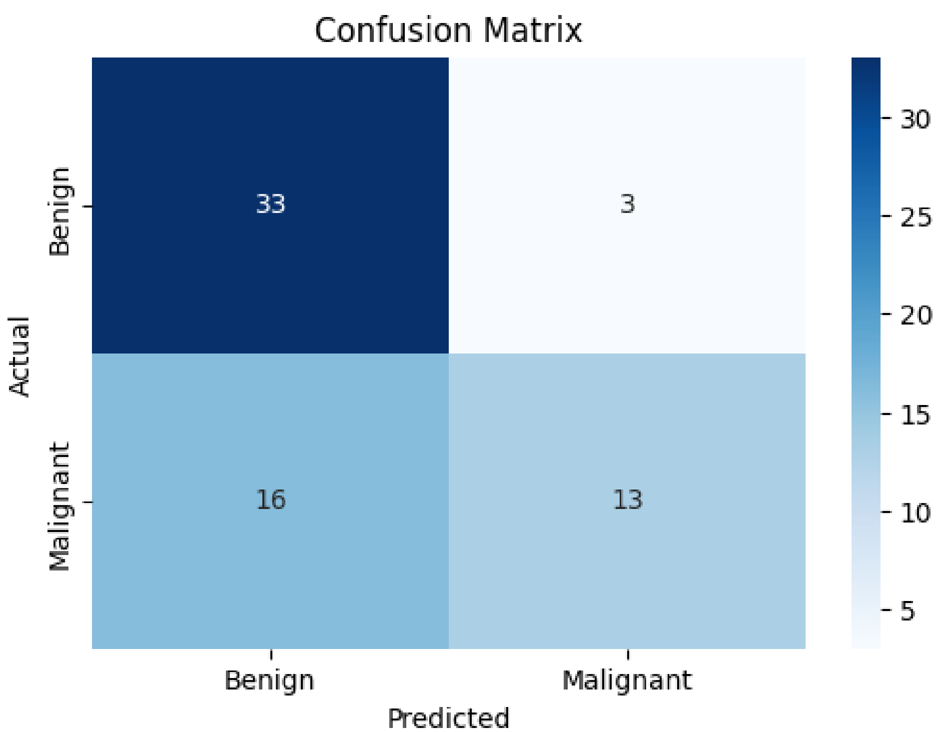
```
100%|

       | 9/9 [00:32<00:00,  3.63s/it]

Epoch 1/5, Train Loss: 0.6700, Train Accuracy: 0.7054, Test Loss: 1.9772, Test
Accuracy: 0.6154

100%|

       | 9/9 [00:34<00:00,  3.81s/it]

Epoch 2/5, Train Loss: 0.3094, Train Accuracy: 0.8488, Test Loss: 0.7036, Test
Accuracy: 0.6154

100%|

       | 9/9 [00:32<00:00,  3.65s/it]
```

```
Epoch 3/5, Train Loss: 0.1361, Train Accuracy: 0.9457, Test Loss: 0.8677, Test
Accuracy: 0.8000

100%|

        | 9/9 [00:33<00:00,  3.70s/it]

Epoch 4/5, Train Loss: 0.2053, Train Accuracy: 0.9922, Test Loss: 1.4933, Test
Accuracy: 0.8462

100%|

        | 9/9 [00:32<00:00,  3.64s/it]

Epoch 5/5, Train Loss: 0.1180, Train Accuracy: 0.9535, Test Loss: 2.0071, Test
Accuracy: 0.7077
```






```
Accuracy: 0.7077, Precision: 0.8125, Recall: 0.4483, F1 Score: 0.5778, MCC:
0.4211
```



[30]:

```python
#IMPORTING REQUIRED LIBRARIES
import torch
from PIL import Image
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# DEFINING THE FUNCTION TO PREDICT IMAGE LABEL
def predict_image(model, image_path):
    ####STEP-1 LOADING THE IMAGE USING PIL
    img = Image.open(image_path)

    ##### STEP-2 IMAGE PREPROCESSING
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
    ])

    img_tensor = transform(img).unsqueeze(0)

    ####STEP-3
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    img_tensor = img_tensor.to(device)

    #####STEP-4 SETTING THE MODEL IN EVALUATION MODE
    model.eval()

    #####STEP-5 MAKE PREDICTIONS
    with torch.no_grad():
        output = model(img_tensor)
        _, predicted_class = torch.max(output, 1)

    #####STEP-6 MAP THE PREDICTED INDEX TO THE ACTUAL CLASS LABEL
    class_names = ['Benign', 'Malignant']
    predicted_label = class_names[predicted_class.item()]

    return predicted_label

####DEFINE THE FUNCTION TO DIPLAY THE IMAGE WITH LABEL
def display_image_with_label(image_path, predicted_label, actual_label):
    img = mpimg.imread(image_path)
    plt.imshow(img)
    plt.title(f'Predicted: {predicted_label}, Actual: {actual_label}')
    plt.axis('off')
    plt.show()
```

```python
#10 IMAGES
image_paths = [
    '/Users/krutikadeshmukh/Desktop/testing images/beingn 5.jpeg',
    '/Users/krutikadeshmukh/Desktop/testing images/malignant.jpeg',
    '/Users/krutikadeshmukh/Desktop/testing images/benign 6.jpeg',
    '/Users/krutikadeshmukh/Desktop/testing images/benign image3.jpeg',
    '/Users/krutikadeshmukh/Desktop/testing images/benignimage 2.jpeg',
    '/Users/krutikadeshmukh/Desktop/testing images/malignant3.jpeg',
    '/Users/krutikadeshmukh/Desktop/testing images/benignimage1.jpeg',
    '/Users/krutikadeshmukh/Desktop/testing images/malignant 1.jpeg',
    '/Users/krutikadeshmukh/Desktop/testing images/malignant 4.jpg',
    '/Users/krutikadeshmukh/Desktop/testing images/malignant2.jpeg'
]

#ACTUAL LABELS FOR THE IMAGES
actual_labels = ['Benign', 'Malignant', 'Benign', 'Benign', 'Benign',
                 'Malignant', 'Benign', 'Malignant', 'Malignant', 'Malignant']

#VARIABLES TO TRACK CORRECT PREDICTIONS
correct_predictions = 0

#LOOP THROUGH IMAGES AND PREDICT
for i, image_path in enumerate(image_paths):
    predicted_label = predict_image(model, image_path)
    actual_label = actual_labels[i]

    #DISPLAYING THE IMAGE WITH PREDICTED LABEL AND ACTUAL LABEL
    display_image_with_label(image_path, predicted_label, actual_label)

    if predicted_label == actual_label:
        print(f"Prediction for image {i+1} is correct!")
        correct_predictions += 1
    else:
        print(f"Prediction for image {i+1} is incorrect!")

### DISPLAYING TOTAL NUMBER OF CORRECT PREDICTIONS
print(f"\nTotal correct predictions: {correct_predictions}/{len(image_paths)}")
```

Predicted: Malignant, Actual: Benign

Prediction for image 1 is incorrect!

Predicted: Benign, Actual: Malignant

Prediction for image 2 is incorrect!

Prediction for image 3 is correct!

Predicted: Malignant, Actual: Benign

```
Prediction for image 4 is incorrect!
```

Predicted: Malignant, Actual: Benign

```
Prediction for image 5 is incorrect!
```

Predicted: Benign, Actual: Malignant

```
Prediction for image 6 is incorrect!
```

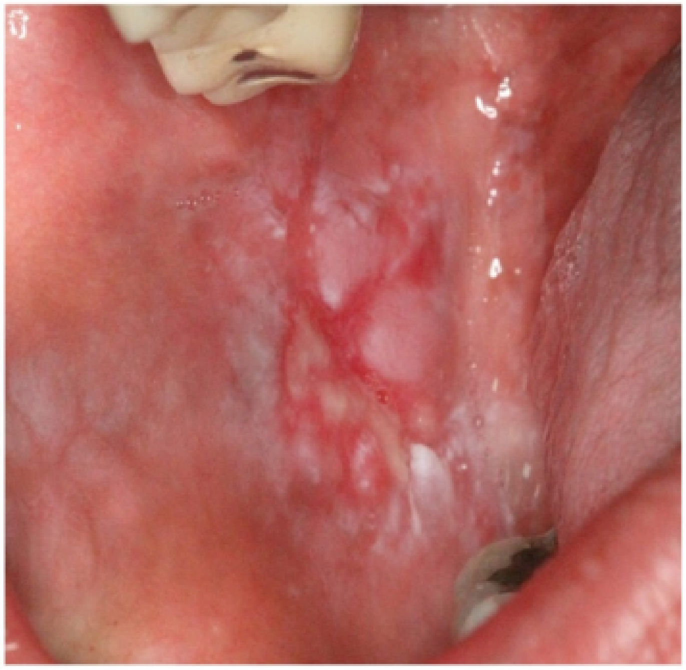
Predicted: Malignant, Actual: Benign

```
Prediction for image 7 is incorrect!
```

Predicted: Benign, Actual: Malignant

```
Prediction for image 8 is incorrect!
```

Predicted: Benign, Actual: Malignant



Prediction for image 9 is incorrect!

Predicted: Benign, Actual: Malignant

```
Prediction for image 10 is incorrect!

Total correct predictions: 1/10
```

[ ]:
```

```