untitled17

October 18, 2024

[36]:
```
import torch
import torch.optim as optim
import torch.nn as nn
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader, random_split
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
```

[37]:
```
#### STEP-1: DEFINING THE DATA PATH
data_dir = "/Users/krutikadeshmukh/Downloads/Oral Images Dataset 2/
↪original_data"  # Original data
```

[38]:
```
##### DATA AUGMENTATION AND PREPROCESSING FOR TRAINING IMAGES
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])
])
```

[39]:
```
### BASIC PREPROCESSING FOR TESTING IMAGES
test_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
])
```

[40]:
```
## STEP-2: LOADING THE ORIGINAL DATASET WITH TRAIN AND TEST TRANSFORMS
original_dataset = datasets.ImageFolder(data_dir, transform=train_transforms)
```

[41]:
```
### STEP-3: SPLITTING THE DATA INTO TRAINING (80%) AND TESTING (20%) SETS
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])
```

[42]:
```
###### APPLYING TEST TRANSFORM TO TEST DATASET
test_dataset.dataset.transform = test_transforms
```

[43]:
```
###### STEP-4: LOADING DATASETS INTO DATA LOADERS
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

[44]:
```
###### STEP-5: LOADING THE VGG19 MODEL
model = models.vgg19(pretrained=False)
```

[45]:
```
##### LOADING THE PRETRAINED WEIGHTS
pth_file_path = "/Users/krutikadeshmukh/Downloads/vgg19-dcbb9e9d.pth"
model.load_state_dict(torch.load(pth_file_path, map_location=torch.
↪device('cpu')))
```

```
/var/folders/qm/w3fd9xt10b90v_xr6kfn5kkh0000gn/T/ipykernel_4459/3150985499.py:3:
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.load_state_dict(torch.load(pth_file_path,
map_location=torch.device('cpu')))
```

[45]:
```
<All keys matched successfully>
```

[46]:
```
###### STEP-6: MODIFYING THE CLASSIFIER FOR BINARY CLASSIFICATION
model.classifier[6] = nn.Linear(in_features=4096, out_features=2)
```

[47]:
```
###### STEP-7: FREEZING ALL THE LAYERS EXCEPT THE LAST THREE
for param in model.features.parameters():
    param.requires_grad = False
```

[48]:
```
#### UNFREEZE THE LAST THREE LAYERS OF THE CLASSIFIER
for param in model.classifier[:4].parameters():
    param.requires_grad = False
for param in model.classifier[4:].parameters():
    param.requires_grad = True
```

[49]:
```
### STEP-8: MOVING MODEL TO GPU IF AVAILABLE
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)
```

[50]:
```
### STEP-9: DEFINING THE LOSS FUNCTION AND OPTIMIZER
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()),␣
 ↪lr=0.001)
```

[51]:
```
## STEP-10: TRAINING THE MODEL AND TRACKING METRICS
def train_model(model, criterion, optimizer, train_loader, test_loader,␣
 ↪num_epochs=10):
    train_losses, test_losses = [], []
    train_accuracies, test_accuracies = [], []

    for epoch in range(num_epochs):
        # Training phase
        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()
```

[52]:

```python
## STEP-11: EVALUATING THE MODEL AND PLOTTING CONFUSION MATRIX WITH METRICS
def plot_confusion_matrix_and_metrics(model, test_loader):
    model.eval()
    all_preds = []
    all_labels = []
    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)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
            correct_test += (preds == labels).sum().item()
            total_test += labels.size(0)

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

    # PLOTTING 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()
```

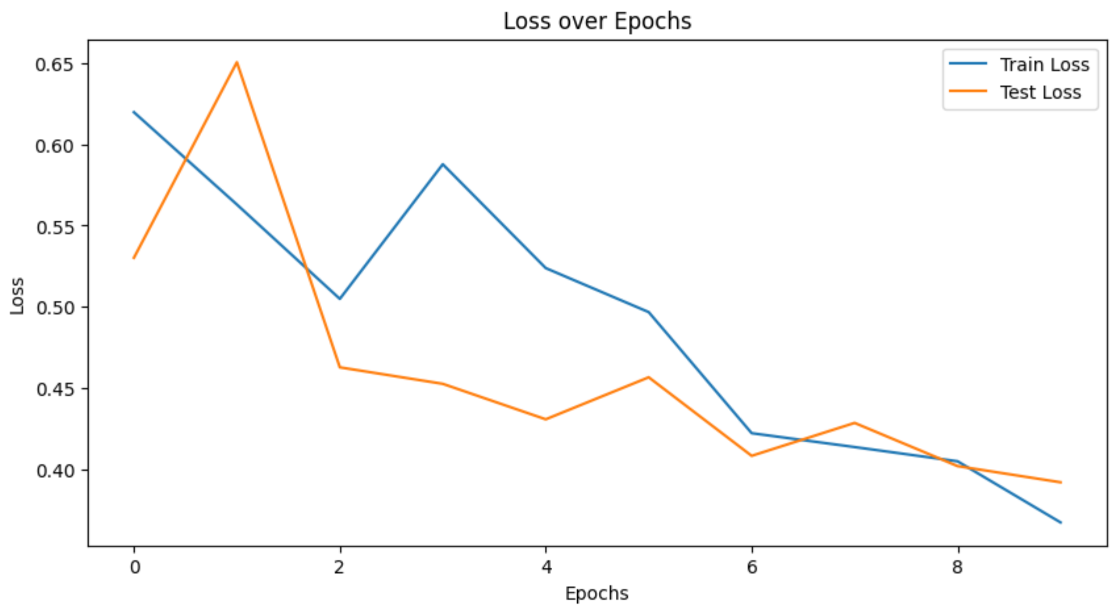
[53]:

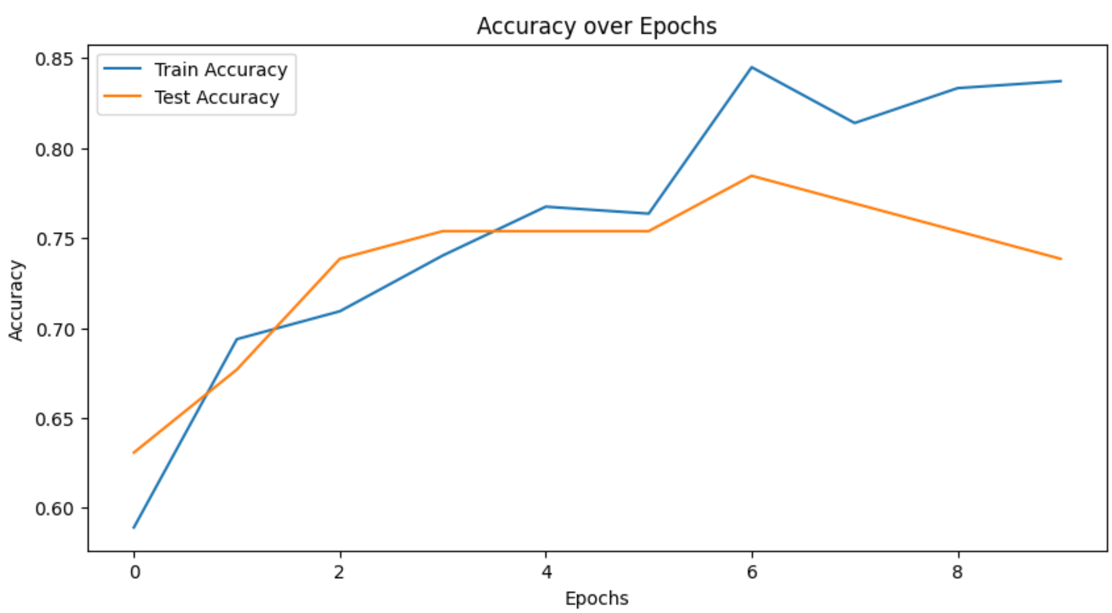
```python
###### STEP-13: RUN TRAINING AND THEN PLOT LOSS/ACCURACY AND CONFUSION MATRIX
# Run the training first
train_model(model, criterion, optimizer, train_loader, test_loader,␣
 ↪num_epochs=10)
```

```
100%|

       | 9/9 [01:03<00:00,  7.00s/it]
```

```
Epoch 1/10, Train Loss: 0.6196, Train Accuracy: 0.5891, Test Loss: 0.5302, Test
Accuracy: 0.6308

100%|

       | 9/9 [01:02<00:00,  6.90s/it]

Epoch 2/10, Train Loss: 0.5629, Train Accuracy: 0.6938, Test Loss: 0.6503, Test
Accuracy: 0.6769

100%|

       | 9/9 [01:00<00:00,  6.77s/it]

Epoch 3/10, Train Loss: 0.5050, Train Accuracy: 0.7093, Test Loss: 0.4629, Test
Accuracy: 0.7385

100%|

       | 9/9 [01:01<00:00,  6.83s/it]

Epoch 4/10, Train Loss: 0.5876, Train Accuracy: 0.7403, Test Loss: 0.4528, Test
Accuracy: 0.7538

100%|

       | 9/9 [01:01<00:00,  6.81s/it]

Epoch 5/10, Train Loss: 0.5238, Train Accuracy: 0.7674, Test Loss: 0.4310, Test
Accuracy: 0.7538

100%|

       | 9/9 [00:59<00:00,  6.61s/it]

Epoch 6/10, Train Loss: 0.4968, Train Accuracy: 0.7636, Test Loss: 0.4568, Test
Accuracy: 0.7538

100%|

       | 9/9 [01:00<00:00,  6.67s/it]

Epoch 7/10, Train Loss: 0.4225, Train Accuracy: 0.8450, Test Loss: 0.4085, Test
Accuracy: 0.7846

100%|

       | 9/9 [01:04<00:00,  7.17s/it]

Epoch 8/10, Train Loss: 0.4139, Train Accuracy: 0.8140, Test Loss: 0.4288, Test
Accuracy: 0.7692

100%|

       | 9/9 [01:03<00:00,  7.04s/it]
```

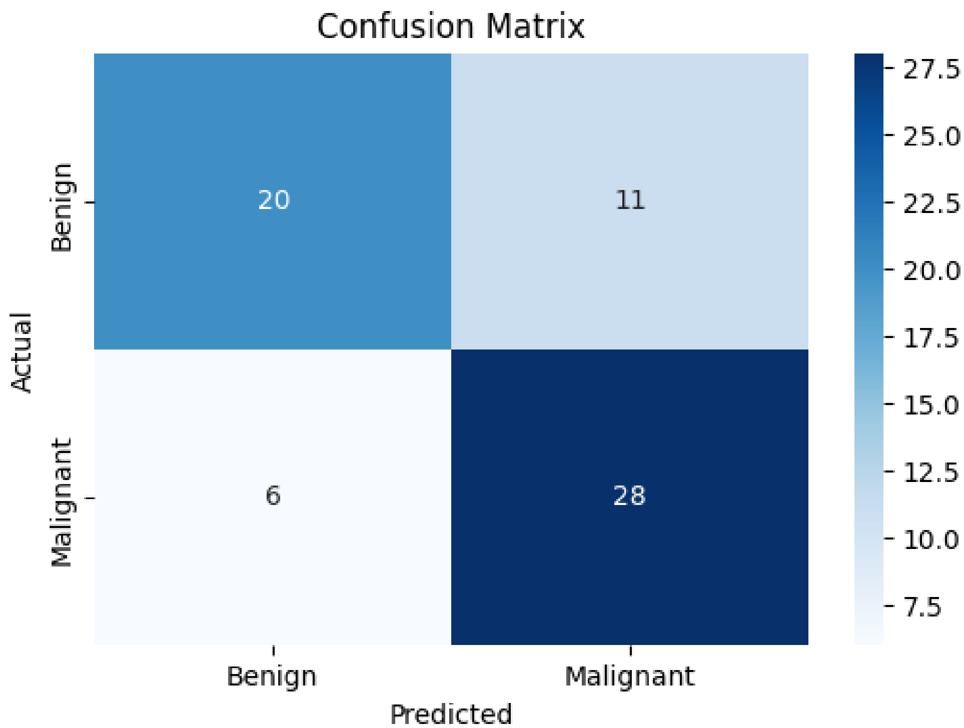
```
Epoch 9/10, Train Loss: 0.4052, Train Accuracy: 0.8333, Test Loss: 0.4022, Test
Accuracy: 0.7538

100%|

       | 9/9 [01:00<00:00,  6.67s/it]

Epoch 10/10, Train Loss: 0.3676, Train Accuracy: 0.8372, Test Loss: 0.3923, Test
Accuracy: 0.7385
```





```
[54]: # PLOT CONFUSION MATRIX AND METRICS ONCE AFTER TRAINING IS COMPLETE
plot_confusion_matrix_and_metrics(model, test_loader)
```

```
Accuracy: 0.7385, Precision: 0.7179, Recall: 0.8235, F1 Score: 0.7671, MCC:
0.4778
```




```
[55]: #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(),
```

```python
        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()

#10 IMAGES
image_paths = [
    '/Users/krutikadeshmukh/Desktop/testing images/beingn 5.jpeg',
    '/Users/krutikadeshmukh/Desktop/testing images/malignant.jpeg',
    '/Users/krutikadeshmukh/Desktop/testing images/benign0.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/malinant5.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', 'Malignant', '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: Malignant, Actual: Malignant

```
Prediction for image 2 is correct!
```

Predicted: Benign, Actual: Benign

```
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: Malignant, Actual: Malignant

```
Prediction for image 6 is correct!
```

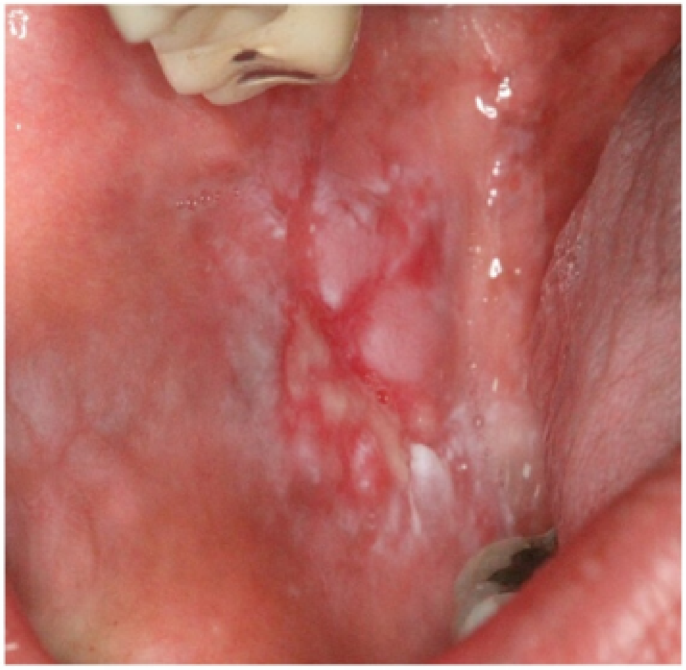
Predicted: Benign, Actual: Malignant

Prediction for image 7 is incorrect!

Predicted: Malignant, Actual: Malignant

Prediction for image 8 is correct!

Predicted: Malignant, Actual: Malignant



Prediction for image 9 is correct!

Predicted: Benign, Actual: Malignant

```
Prediction for image 10 is incorrect!

Total correct predictions: 5/10
```

[ ]:

```

```