



```
In [ ]: ! conda list
```

```
In [ ]: ! pip install torch torchvision
```

```
In [ ]: ! pip install scikit-learn
```

```
In [ ]: ! pip install matplotlib
```

```
In [5]: import os

def delete_files(directory, num_files):
    try:
        # List all files in directory
        files = [f for f in os.listdir(directory) if os.path.isfile(os.path.join(directory, f))]

        # Sort files by modification time (oldest first)
        files.sort(key=lambda x: os.path.getmtime(os.path.join(directory, x)))

        # Select only the number of files to delete
        files_to_delete = files[:num_files]

        for file in files_to_delete:
            file_path = os.path.join(directory, file)
            os.remove(file_path)
            print(f"Deleted: {file_path}")

        print(f"\nDeleted {len(files_to_delete)} files from {directory}")

    except Exception as e:
        print(f"Error: {e}")
```

```
In [6]: # Example usage:
# change these values before running
directory_path = "./ThermalResFolder/Healthy" # directory where files exist
number_of_files = 2118                        # number of files to delete

delete_files(directory_path, number_of_files)
```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Deleted: ./ThermalResFolder/Healthy\T0108.1.1.D.2012-11-28.14.png
Deleted: ./ThermalResFolder/Healthy\T0108.1.1.D.2012-11-28.15.png
Deleted: ./ThermalResFolder/Healthy\T0108.1.1.D.2012-11-28.16.png
Deleted: ./ThermalResFolder/Healthy\T0108.1.1.D.2012-11-28.17.png
Deleted: ./ThermalResFolder/Healthy\T0108.1.1.D.2012-11-28.18.png
Deleted: ./ThermalResFolder/Healthy\T0108.1.1.D.2012-11-28.19.png
Deleted: ./ThermalResFolder/Healthy\T0108.1.1.S.2012-11-28.00.png
Deleted: ./ThermalResFolder/Healthy\T0108.1.2.S.2012-11-28.00.png
Deleted: ./ThermalResFolder/Healthy\T0108.1.3.D.2012-11-28.00.png
Deleted: ./ThermalResFolder/Healthy\T0108.1.3.S.2012-11-28.00.png
Deleted: ./ThermalResFolder/Healthy\T0108.1.4.S.2012-11-28.00.png
Deleted: ./ThermalResFolder/Healthy\T0108.1.5.D.2012-11-28.00.png

Deleted 2118 files from ./ThermalResFolder/Healthy

```
In [7]: import os

def count_files(directory):
    file_count = 0
    for _, _, files in os.walk(directory):
        file_count += len(files)
    return file_count
```

```
In [8]: # Example usage
directory_path = "./ThermalResFolder/Healthy" # change this path
print(f"Total number of files in '{directory_path}': {count_files(directory_path)}
```

Total number of files in './ThermalResFolder/Healthy': 1068

```
In [9]: # Example usage
directory_path = "./ThermalResFolder/sick" # change this path
print(f"Total number of files in '{directory_path}': {count_files(directory_path)}
```

Total number of files in './ThermalResFolder/sick': 1068

```
In [10]: import os
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
from torch.utils.data import DataLoader, random_split
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import numpy as np

# Paths
data_dir = './ThermalResFolder' # Replace with your dataset root folder path

# Parameters
batch_size = 32
num_epochs = 10
learning_rate = 0.001
```

```

validation_split = 0.2
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Transformations (resize, tensor, normalize as ResNet requires)
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], # ImageNet mean/std for
                        std=[0.229, 0.224, 0.225])
])

# Load dataset
full_dataset = datasets.ImageFolder(root=data_dir, transform=transform)

# Split into train and validation sets
val_size = int(len(full_dataset) * validation_split)
train_size = len(full_dataset) - val_size
train_dataset, val_dataset = random_split(full_dataset, [train_size, val_size])

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True,
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num

# Load pretrained ResNet18 and replace final layer for 2 classes
model = models.resnet18(pretrained=True)
model.fc = nn.Linear(model.fc.in_features, 2)
model = model.to(device)

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

# To record training history
train_losses = []
val_losses = []
val_aucs = []

# Training and validation loop
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0

    for inputs, labels in 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()

    epoch_train_loss = running_loss / len(train_loader)

```

```

train_losses.append(epoch_train_loss)

# Validation
model.eval()
val_loss = 0.0
all_labels = []
all_probs = []

with torch.no_grad():
    for val_inputs, val_labels in val_loader:
        val_inputs, val_labels = val_inputs.to(device), val_labels.to(device)
        val_outputs = model(val_inputs)
        loss = criterion(val_outputs, val_labels)
        val_loss += loss.item()

        probs = torch.softmax(val_outputs, dim=1)[:, 1] # Probability for
        all_probs.extend(probs.cpu().numpy())
        all_labels.extend(val_labels.cpu().numpy())

epoch_val_loss = val_loss / len(val_loader)
val_losses.append(epoch_val_loss)

# Calculate AUC
auc_score = roc_auc_score(all_labels, all_probs)
val_aucs.append(auc_score)

print(f"Epoch {epoch+1}/{num_epochs}: "
      f"Train Loss: {epoch_train_loss:.4f}, "
      f"Val Loss: {epoch_val_loss:.4f}, "
      f"Val AUC: {auc_score:.4f}")

```

d:\Gourav\Anaconda\envs\Thermal\lib\site-packages\torchvision\models_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.

warnings.warn(
d:\Gourav\Anaconda\envs\Thermal\lib\site-packages\torchvision\models_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use `weights=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
warnings.warn(msg)

Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to C:\Users\Sree Ram\.cache\torch\hub\checkpoints\resnet18-f37072fd.pth

100.0%

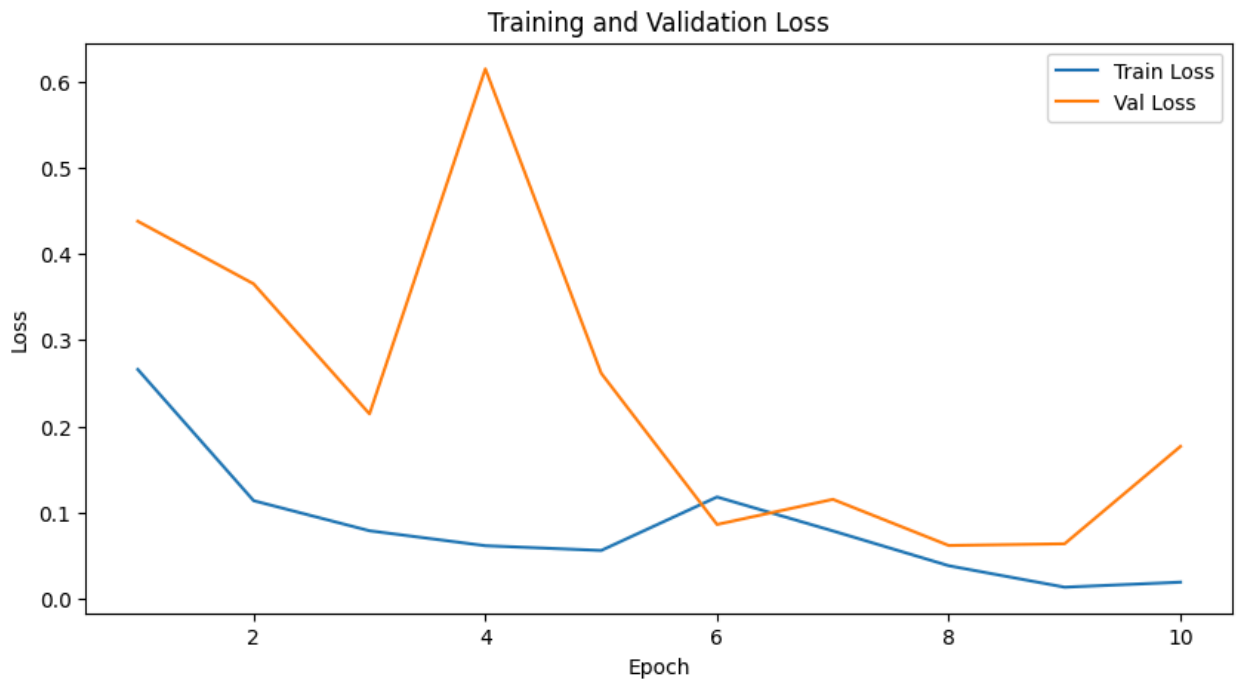
```

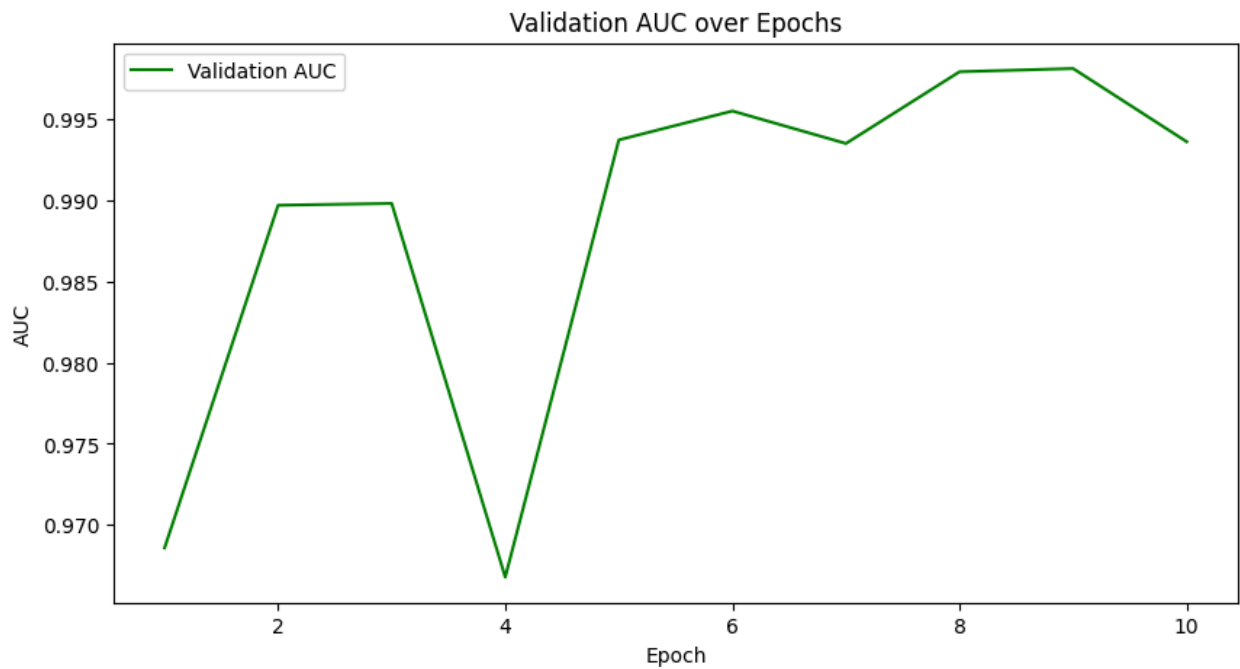
Epoch 1/10: Train Loss: 0.2659, Val Loss: 0.4377, Val AUC: 0.9686
Epoch 2/10: Train Loss: 0.1137, Val Loss: 0.3652, Val AUC: 0.9897
Epoch 3/10: Train Loss: 0.0789, Val Loss: 0.2144, Val AUC: 0.9898
Epoch 4/10: Train Loss: 0.0615, Val Loss: 0.6144, Val AUC: 0.9668
Epoch 5/10: Train Loss: 0.0560, Val Loss: 0.2615, Val AUC: 0.9937
Epoch 6/10: Train Loss: 0.1180, Val Loss: 0.0862, Val AUC: 0.9955
Epoch 7/10: Train Loss: 0.0787, Val Loss: 0.1153, Val AUC: 0.9935
Epoch 8/10: Train Loss: 0.0383, Val Loss: 0.0618, Val AUC: 0.9979
Epoch 9/10: Train Loss: 0.0135, Val Loss: 0.0638, Val AUC: 0.9981
Epoch 10/10: Train Loss: 0.0193, Val Loss: 0.1767, Val AUC: 0.9936

```

```
In [11]: # Plot training and validation loss curves
plt.figure(figsize=(10,5))
plt.plot(range(1, num_epochs+1), train_losses, label='Train Loss')
plt.plot(range(1, num_epochs+1), val_losses, label='Val Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()

# Plot AUC score curve
plt.figure(figsize=(10,5))
plt.plot(range(1, num_epochs+1), val_aucs, label='Validation AUC', color='green')
plt.xlabel('Epoch')
plt.ylabel('AUC')
plt.title('Validation AUC over Epochs')
plt.legend()
plt.show()
```





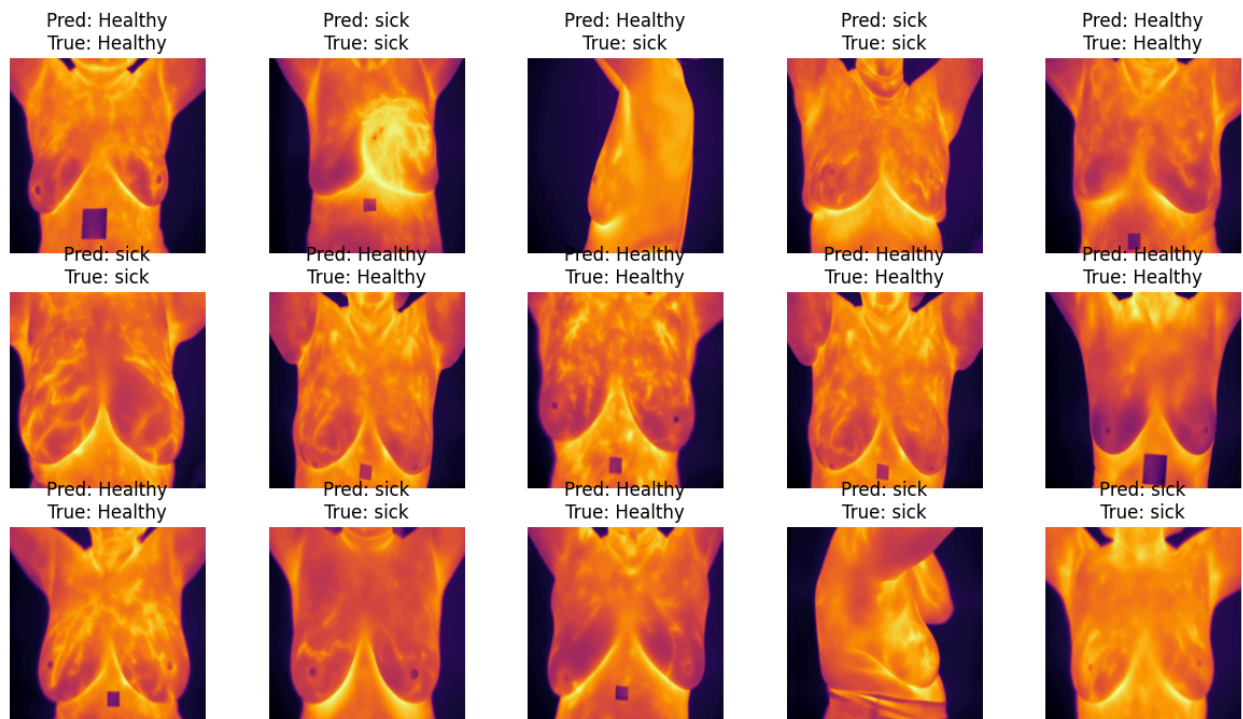
```
In [12]: # Basic test visualization: show some validation images with predicted labels
def imshow(inp, title=None):
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.axis('off')

model.eval()
images_shown = 0
class_names = full_dataset.classes

plt.figure(figsize=(15, 8))
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs = inputs.to(device)
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)

        for i in range(inputs.size()[0]):
            images_shown += 1
            plt.subplot(3, 5, images_shown)
            imshow(inputs.cpu().data[i], title=f"Pred: {class_names[preds[i]]}")
            if images_shown == 15:
                break
        if images_shown == 15:
            break

plt.show()
```



```
In [13]: from sklearn.metrics import classification_report, confusion_matrix, f1_score,
import seaborn as sns
```

```
# Put the model in evaluation mode
model.eval()

all_labels = []
all_preds = []

with torch.no_grad():
    for inputs, labels in val_loader:
        inputs = inputs.to(device)
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Calculate and print accuracy, precision, recall, F1
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')

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

# Full classification report
target_names = full_dataset.classes
print("\nClassification Report:")
```

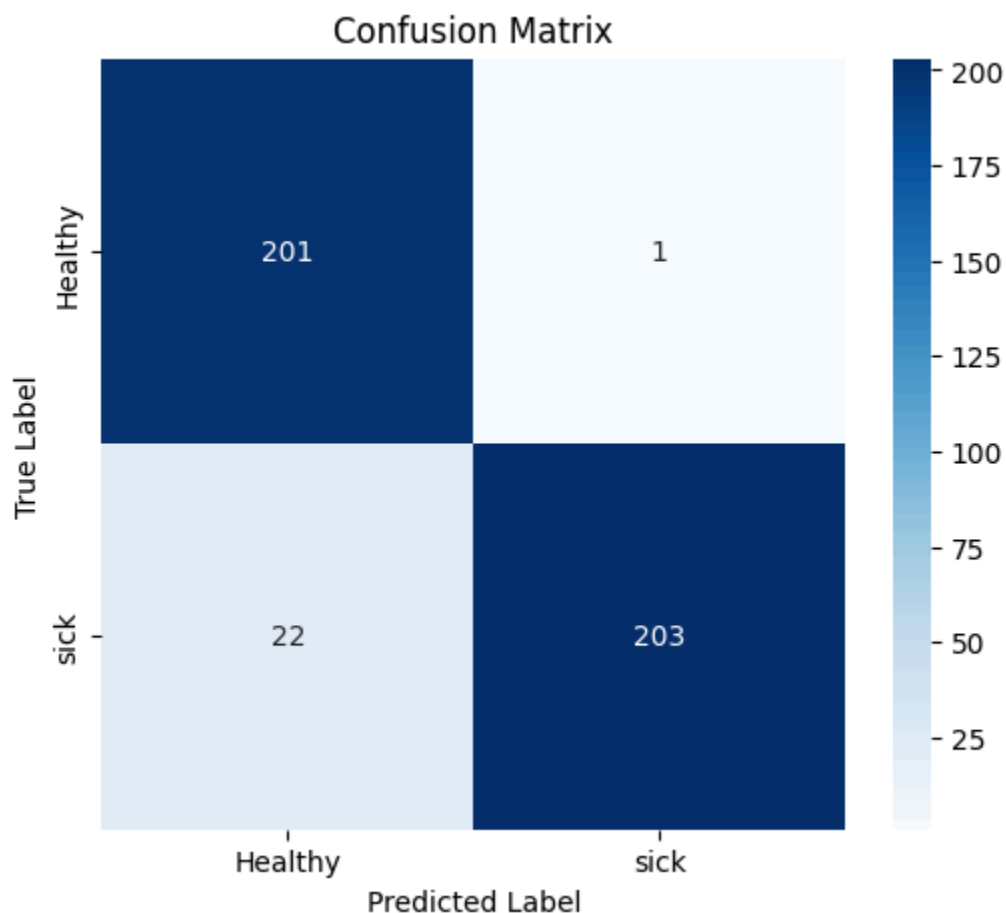
```
print(classification_report(all_labels, all_preds, target_names=target_names))

# Confusion matrix
cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=target_names, y
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

Validation Accuracy: 0.9461
Validation Precision: 0.9951
Validation Recall: 0.9022
Validation F1 Score: 0.9464

Classification Report:

	precision	recall	f1-score	support
Healthy	0.90	1.00	0.95	202
sick	1.00	0.90	0.95	225
accuracy			0.95	427
macro avg	0.95	0.95	0.95	427
weighted avg	0.95	0.95	0.95	427



```
In [14]: # Save the entire model  
torch.save(model, "mammo_ml.pth")
```

```
In [ ]: # Load later  
# model = torch.load("full_model.pth")  
# model.eval()
```