Week_6_First_Modeling

October 14, 2024

[2]:
```python
# Standard Libraries
import io
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Deep Learning and PyTorch
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader

# Image Processing
from PIL import Image
from torchvision import transforms, models

# File Handling
import h5py

# Metrics and Evaluation
from sklearn.metrics import classification_report, roc_auc_score, roc_curve, auc

# Progress Visualization
from tqdm import tqdm
```

0.1 Create Custom Dataset

[3]:
```python
class MultiInputDataset(Dataset):
    def __init__(self, hdf5_file, csv_file, transform=None):
        # Open the HDF5 file with error handling
        try:
            self.hdf5_file = h5py.File(hdf5_file, 'r')  # Read-only mode
        except Exception as e:
            raise IOError(f"Could not open HDF5 file: {hdf5_file}. Error: {e}")

        # Read the CSV file containing image labels and additional features
```

```
        try:
            self.labels_df = pd.read_csv(csv_file)
        except Exception as e:
            raise IOError(f"Could not read CSV file: {csv_file}. Error: {e}")

        # Ensure that all image IDs from the CSV are present in the HDF5 file
        self.image_ids = self.labels_df['isic_id'].values
        for image_id in self.image_ids:
            if str(image_id) not in self.hdf5_file.keys():
                raise ValueError(f"Image id {image_id} not found in HDF5 file.")

        # Store any transformations to be applied to the images
        self.transform = transform

    def __len__(self):
        # Return the total number of samples in the dataset
        return len(self.labels_df)

    def __getitem__(self, idx):
        # Get the image ID from the CSV file based on index
        image_id = str(self.labels_df.iloc[idx]['isic_id'])

        # Load the image data from the HDF5 file
        image_bytes = self.hdf5_file[image_id][()]

        # Convert the image bytes to a PIL Image
        image = Image.open(io.BytesIO(image_bytes))

        # Apply any specified transformations to the image
        if self.transform:
            image = self.transform(image)

        # Retrieve the label
        label = torch.tensor(self.labels_df.iloc[idx]['target'], dtype=torch.
↪long)  # Adjust dtype if needed

        # Retrieve other features, excluding 'isic_id' and 'target'
        other_variables = self.labels_df.iloc[idx].drop(['isic_id', 'target']).
↪values.astype(float)

        # Convert other variables (metadata) to a tensor
        metadata_tensor = torch.tensor(other_variables, dtype=torch.float32)

        # Return the image, metadata, and label
        return image, metadata_tensor, label
```

```
[4]: # Define any necessary transformations for the image dataset


# Transformations for training set (with data augmentation)
train_transform = transforms.Compose([
    transforms.Resize((128, 128)),  # Resize to 225x225
    transforms.RandomResizedCrop(128, scale=(0.8, 1.0)),  # Random crop to␣
 ↪225x225 with scale
    transforms.RandomRotation(10),  # Randomly rotate images by 10 degrees
    transforms.ToTensor(),  # Convert image to PyTorch tensor
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Transformations for validation/test set (no data augmentation)
normal_transform = transforms.Compose([
    transforms.Resize((128, 128)),  # Ensure the image is resized to 225x225
    transforms.ToTensor(),  # Convert image to PyTorch tensor
])
```

0.2 Train DataLoader

```
[5]: # Initialize the dataset
train_dataset = MultiInputDataset(hdf5_file='../data/raw/train_images.hdf5',␣
 ↪csv_file='../data/processed/processed-train-metadata1.csv',␣
 ↪transform=normal_transform)
val_dataset = MultiInputDataset(hdf5_file='../data/raw/validation_image.hdf5',␣
 ↪csv_file='../data/processed/processed-validation-metadata1.csv',␣
 ↪transform=normal_transform)
# Create a DataLoader
train_dataloader = DataLoader(train_dataset,  batch_size=64, shuffle=True)
val_dataloader = DataLoader(val_dataset,  batch_size=64, shuffle=True)
```

```
[6]: device = "cuda" if torch.cuda.is_available() else "cpu"
```

0.3 Model Building

```
[7]: class CustomImageFeatureCNN2(nn.Module):
    def __init__(self, feature_input_size, input_image_size=(128, 128)):
        super(CustomImageFeatureCNN2, self).__init__()

        # Image CNN with Batch Normalization
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3,␣
 ↪padding=1)
        self.bn1 = nn.BatchNorm2d(32)  # Batch normalization after conv1

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
```

```python
        self.bn2 = nn.BatchNorm2d(64)  # Batch normalization after conv2

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)  # Batch normalization after conv3

        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)  # 2x2 Max pooling

        # Dynamically calculate the flattened size of the feature map
        self.flattened_size = self._get_flattened_size(input_image_size)

        # Fully connected layer after the CNN layers
        self.fc_image = nn.Linear(self.flattened_size, 512)

        # Fully connected layer for metadata (feature data)
        self.fc_metadata = nn.Linear(feature_input_size, 128)

        # Dropout layer to prevent overfitting
        self.dropout = nn.Dropout(0.5)  # 50% dropout

        # Final fully connected layer for binary classification (combined image + feature input)
        self.fc_combined = nn.Linear(512 + 128, 1)  # Change 2 to 1 for binary classification

    def _get_flattened_size(self, input_image_size):
        # Forward pass a dummy image to get the size of the flattened features
        dummy_image = torch.zeros(1, 3, *input_image_size)  # Batch size of 1, 3 channels (RGB), and input size
        x = self.pool(F.relu(self.bn1(self.conv1(dummy_image))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.pool(F.relu(self.bn3(self.conv3(x))))
        return x.view(-1).shape[0]  # Flatten and return the size

    def forward(self, image, metadata):
        # Forward pass for the image through the CNN
        x = self.pool(F.relu(self.bn1(self.conv1(image))))  # Conv layer 1 with ReLU, BatchNorm, MaxPool
        x = self.pool(F.relu(self.bn2(self.conv2(x))))  # Conv layer 2 with ReLU, BatchNorm, MaxPool
        x = self.pool(F.relu(self.bn3(self.conv3(x))))  # Conv layer 3 with ReLU, BatchNorm, MaxPool

        # Flatten the feature map to feed into fully connected layer
        x = x.view(x.size(0), -1)  # Flatten feature maps into a 1D vector
        image_features = F.relu(self.fc_image(x))
```

```
        # Process metadata (feature data)
        metadata_features = F.relu(self.fc_metadata(metadata))

        # Ensure the batch sizes are consistent
        assert image_features.shape[0] == metadata_features.shape[0], \
            f"Batch sizes do not match! Image batch size: {image_features.
↪shape[0]}, Metadata batch size: {metadata_features.shape[0]}"

        # Concatenate image features and metadata features
        combined_features = torch.cat((image_features, metadata_features),␣
↪dim=1)

        # Dropout and final classification layer
        combined_features = self.dropout(combined_features)
        output = self.fc_combined(combined_features)

        # If you're using BCELoss, uncomment the next line to apply sigmoid
        output = torch.sigmoid(output)

        return output
```

0.4 Model Training

[8]:

```
# Function to compute partial AUC-above-TPR
def score(solution: np.array, submission: np.array, min_tpr: float = 0.80) ->␣
↪float:
    """
    Compute the partial AUC by focusing on a specific range of true positive␣
↪rates (TPR).

    Args:
        solution (np.array): Ground truth binary labels.
        submission (np.array): Model predictions.
        min_tpr (float): Minimum true positive rate to calculate partial AUC.

    Returns:
        float: The calculated partial AUC.

    Raises:
        ValueError: If the min_tpr is not within a valid range.
    """
    # Rescale the target to handle sklearn limitations and flip the predictions
    v_gt = abs(solution - 1)
    v_pred = -1.0 * submission
    max_fpr = abs(1 - min_tpr)
```

```
    # Compute ROC curve using sklearn
    fpr, tpr, _ = roc_curve(v_gt, v_pred)
    if max_fpr is None or max_fpr == 1:
        return auc(fpr, tpr)
    if max_fpr <= 0 or max_fpr > 1:
        raise ValueError(f"Expected min_tpr in range [0, 1), got: {min_tpr}")

    # Interpolate for partial AUC
    stop = np.searchsorted(fpr, max_fpr, "right")
    x_interp = [fpr[stop - 1], fpr[stop]]
    y_interp = [tpr[stop - 1], tpr[stop]]
    tpr = np.append(tpr[:stop], np.interp(max_fpr, x_interp, y_interp))
    fpr = np.append(fpr[:stop], max_fpr)
    partial_auc = auc(fpr, tpr)

    return partial_auc

# Training and validation loop function
def train_and_validate(
    model: nn.Module,
    train_dataloader: torch.utils.data.DataLoader,
    val_dataloader: torch.utils.data.DataLoader,
    criterion: nn.Module,
    optimizer: torch.optim.Optimizer,
    epochs: int,
    device: torch.device,
    early_stopping_patience: int = 5,
    min_tpr: float = 0.80
) -> nn.Module:
    """
    Train and validate a PyTorch model with early stopping, AUROC, partial AUC,␣
↪and error handling.

    Args:
        model (nn.Module): The model to be trained and validated.
        train_dataloader (torch.utils.data.DataLoader): Dataloader for training␣
↪data.
        val_dataloader (torch.utils.data.DataLoader): Dataloader for validation␣
↪data.
        criterion (nn.Module): Loss function.
        optimizer (torch.optim.Optimizer): Optimizer to update the model.
        epochs (int): Number of training epochs.
        device (torch.device): The device (CPU or GPU) to use.
        early_stopping_patience (int): Early stopping patience.
        min_tpr (float): The minimum true positive rate for calculating partial␣
↪AUC.
```

```
    Returns:
        nn.Module: The trained model.
    """
    # Initialize tracking variables
    best_val_loss = float('inf')
    best_epoch = 0
    train_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []
    early_stopping_counter = 0

    # Start the training and validation loop
    for epoch in range(epochs):
        print(f'Epoch {epoch + 1}/{epochs}')

        # Training phase
        model.train()
        running_train_loss = 0.0
        correct_train = 0
        total_train = 0
        all_train_labels = []
        all_train_probs = []

        progress_bar = tqdm(train_dataloader, desc=f'Training Epoch {epoch + 1}')

        try:
            # Loop through the training batches
            for i, (image, metadata, labels) in enumerate(progress_bar):
                image, metadata, labels = image.to(device), metadata.to(device), labels.float().to(device)
                labels = labels.unsqueeze(1)  # Adjust labels to have the right shape for binary classification

                optimizer.zero_grad()

                # Forward pass
                probs = model(image, metadata)

                if probs.shape != labels.shape:
                    raise ValueError(f"Shape mismatch: Predictions shape {probs.shape} does not match labels shape {labels.shape}")

                # Calculate loss and backpropagate
                loss = criterion(probs, labels)
                loss.backward()
```

```
            optimizer.step()

            # Update running loss
            running_train_loss += loss.item()

            # Store labels and predictions for accuracy calculations
            all_train_labels.extend(labels.cpu().detach().numpy())
            all_train_probs.extend(probs.cpu().detach().numpy())

            # Calculate binary predictions for training accuracy
            predicted_train = (probs >= 0.5).float()
            total_train += labels.size(0)
            correct_train += (predicted_train == labels).sum().item()

            # Update progress bar
            progress_bar.set_postfix(train_loss=running_train_loss / (i +
↪1))

        # Calculate training accuracy and loss
        train_accuracy = 100 * correct_train / total_train
        train_losses.append(running_train_loss / len(train_dataloader))
        train_accuracies.append(train_accuracy)

    except ValueError as ve:
        print(f"Error during training loop: {ve}")
        break

    # Validation phase
    model.eval()
    running_val_loss = 0.0
    correct = 0
    total = 0
    all_labels = []
    all_probs = []

    progress_bar = tqdm(val_dataloader, desc=f'Validating Epoch {epoch +
↪1}')

    with torch.no_grad():
        try:
            # Loop through the validation batches
            for i, (images, metadata, labels) in enumerate(progress_bar):
                images, metadata, labels = images.to(device), metadata.
↪to(device), labels.float().to(device)
                labels = labels.unsqueeze(1)

                probs = model(images, metadata)
```

```
                loss = criterion(probs, labels)
                running_val_loss += loss.item()

                all_labels.extend(labels.cpu().detach().numpy())
                all_probs.extend(probs.cpu().detach().numpy())

                # Calculate binary predictions for validation accuracy
                predicted = (probs >= 0.5).float()
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

                progress_bar.set_postfix(val_loss=running_val_loss / (i +␣
↪1))

            val_accuracy = 100 * correct / total
            val_loss = running_val_loss / len(val_dataloader)
            val_accuracies.append(val_accuracy)
            val_losses.append(val_loss)

            # Calculate AUROC
            try:
                valid_auroc = roc_auc_score(all_labels, all_probs)
            except ValueError as ve:
                print(f"AUROC Calculation Error: {ve}")
                valid_auroc = 0.0

            # Calculate partial AUC-above-TPR
            try:
                partial_auroc = score(np.array(all_labels), np.
↪array(all_probs), min_tpr=min_tpr)
            except ValueError as ve:
                print(f"Partial AUC Calculation Error: {ve}")
                partial_auroc = 0.0

            print(f'Epoch [{epoch + 1}/{epochs}], Train Loss:␣
↪{train_losses[-1]:.4f}, Val Loss: {val_loss:.4f}, '
                  f'Val Accuracy: {val_accuracy:.2f}%, Val AUROC:␣
↪{valid_auroc:.4f}, Partial AUROC: {partial_auroc:.4f}')

            # Early stopping based on validation loss
            if val_loss < best_val_loss:
                best_val_loss = val_loss
                best_epoch = epoch + 1
                early_stopping_counter = 0
                torch.save(model.state_dict(), 'best_model.pth')
            else:
```

```
                early_stopping_counter += 1

            if early_stopping_counter >= early_stopping_patience:
                print(f"Early stopping triggered at epoch {epoch + 1}")
                break

        except Exception as e:
            print(f"Error during validation loop: {e}")
            break

    print(f"Best Epoch: {best_epoch}, Best Validation Loss: {best_val_loss:.
↪4f}")
    print('Training Complete')

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

    # Plot training and validation accuracy
    plt.figure(figsize=(10, 5))
    plt.plot(train_accuracies, label='Train Accuracy')
    plt.plot(val_accuracies, label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy (%)')
    plt.title('Training and Validation Accuracy')
    plt.legend()
    plt.show()

    # Generate classification report
    try:
        print("Classification Report:")
        print(classification_report(all_labels, (np.array(all_probs) >= 0.5).
↪astype(int), target_names=['Class 0', 'Class 1']))
    except Exception as e:
        print(f"Error generating classification report: {e}")

    return model
```

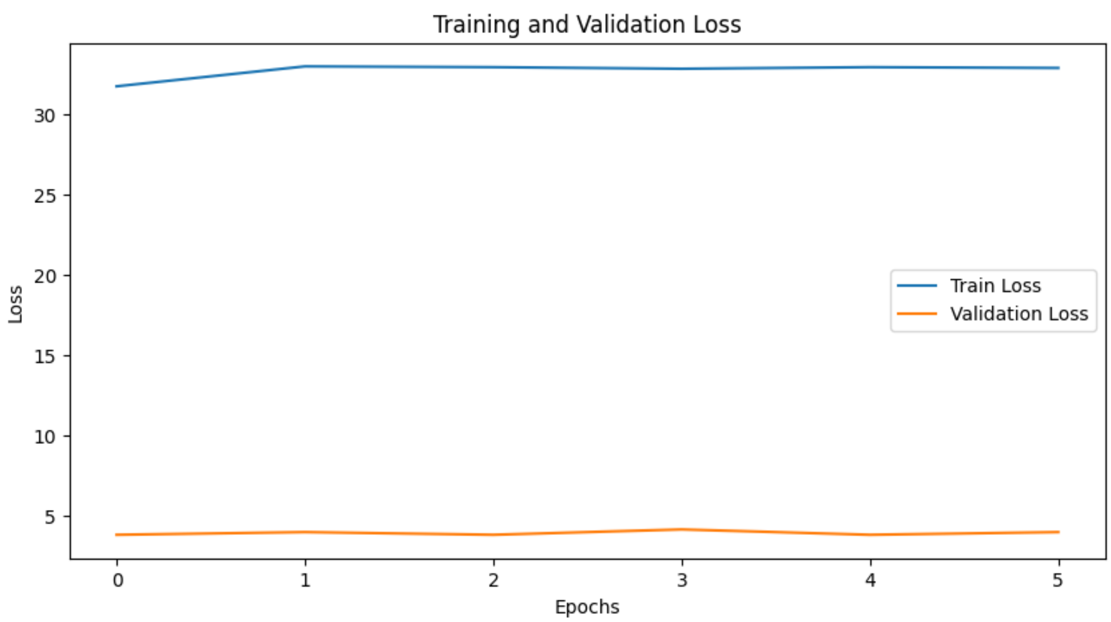
0.5 Model 1

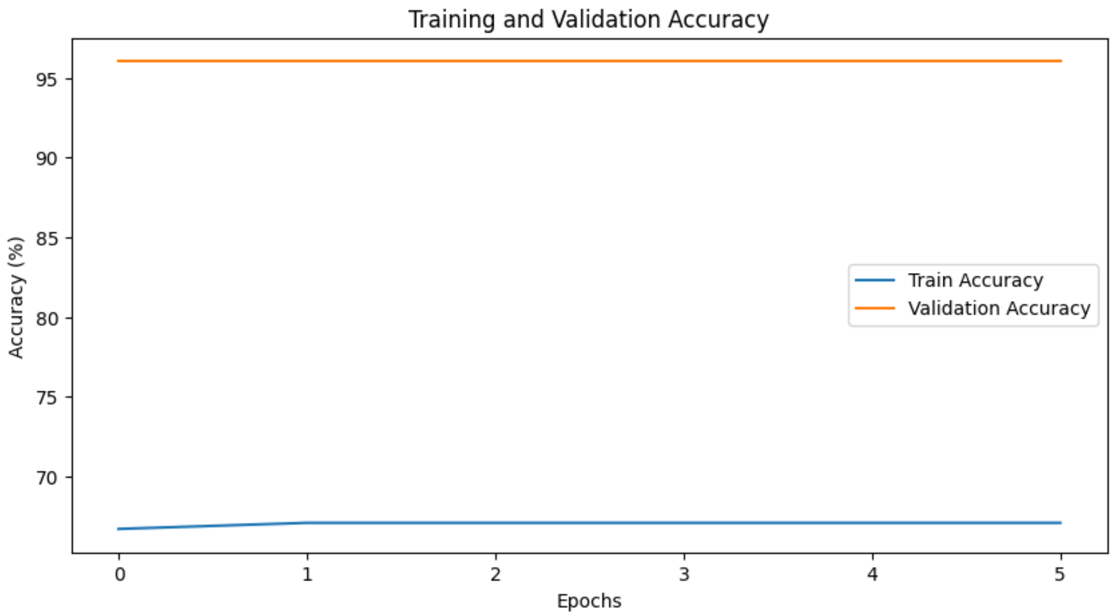
[8]:
```
model1 = CustomImageFeatureCNN2(feature_input_size=9)  # Assuming 9 features␣
 ↪for metadata
model1.to(device)
# Initialize optimizer
optimizer = optim.Adam(model1.parameters(), lr=0.001)
# Define the loss function with the class weights
criterion = nn.BCELoss()  # Binary classification loss
# Set the number of epochs
epochs = 20
```

[9]:
```
train_and_validate(model1,train_dataloader, val_dataloader, criterion,␣
 ↪optimizer, epochs, device )
```

```
Epoch 1/20

Training Epoch 1: 100%|      | 33/33 [01:27<00:00,  2.64s/it,
train_loss=31.7]
Validating Epoch 1: 100%|      | 24/24 [00:24<00:00,  1.02s/it,
val_loss=3.84]

Epoch [1/20], Train Loss: 31.7326, Val Loss: 3.8411, Val Accuracy: 96.04%, Val
AUROC: 0.5000, Partial AUROC: 0.0200
Epoch 2/20

Training Epoch 2: 100%|      | 33/33 [01:20<00:00,  2.45s/it, train_loss=33]
Validating Epoch 2: 100%|      | 24/24 [00:25<00:00,  1.07s/it,
val_loss=4.01]

Epoch [2/20], Train Loss: 32.9726, Val Loss: 4.0075, Val Accuracy: 96.04%, Val
AUROC: 0.5000, Partial AUROC: 0.0200
Epoch 3/20

Training Epoch 3: 100%|      | 33/33 [01:22<00:00,  2.50s/it,
train_loss=32.9]
Validating Epoch 3: 100%|      | 24/24 [00:26<00:00,  1.09s/it,
val_loss=3.84]

Epoch [3/20], Train Loss: 32.9230, Val Loss: 3.8411, Val Accuracy: 96.04%, Val
AUROC: 0.5000, Partial AUROC: 0.0200
Epoch 4/20

Training Epoch 4: 100%|      | 33/33 [01:24<00:00,  2.55s/it,
train_loss=32.8]
Validating Epoch 4: 100%|      | 24/24 [00:26<00:00,  1.10s/it,
val_loss=4.17]

Epoch [4/20], Train Loss: 32.8238, Val Loss: 4.1739, Val Accuracy: 96.04%, Val
AUROC: 0.5000, Partial AUROC: 0.0200
Epoch 5/20
```

```
Training Epoch 5: 100%|      | 33/33 [01:22<00:00,  2.49s/it,
train_loss=32.9]
Validating Epoch 5: 100%|      | 24/24 [00:24<00:00,  1.01s/it,
val_loss=3.84]

Epoch [5/20], Train Loss: 32.9230, Val Loss: 3.8411, Val Accuracy: 96.04%, Val
AUROC: 0.5000, Partial AUROC: 0.0200
Epoch 6/20

Training Epoch 6: 100%|      | 33/33 [01:21<00:00,  2.48s/it,
train_loss=32.9]
Validating Epoch 6: 100%|      | 24/24 [00:23<00:00,  1.00it/s,
val_loss=4.01]

Epoch [6/20], Train Loss: 32.8734, Val Loss: 4.0075, Val Accuracy: 96.04%, Val
AUROC: 0.5000, Partial AUROC: 0.0200
Early stopping triggered at epoch 6
Best Epoch: 1, Best Validation Loss: 3.8411
Training Complete
```






```
Classification Report:
              precision    recall  f1-score   support

     Class 0       0.96      1.00      0.98      1431
     Class 1       0.00      0.00      0.00        59

    accuracy                           0.96      1490
   macro avg       0.48      0.50      0.49      1490
weighted avg       0.92      0.96      0.94      1490
```

```
/opt/tljh/user/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/tljh/user/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/tljh/user/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

[9]:
```
CustomImageFeatureCNN2(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (fc_image): Linear(in_features=32768, out_features=512, bias=True)
  (fc_metadata): Linear(in_features=9, out_features=128, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc_combined): Linear(in_features=640, out_features=1, bias=True)
)
```

0.6 Model 2

[10]:
```
model2 = CustomImageFeatureCNN2(feature_input_size=9)  # Assuming 9 features␣
 ↪for metadata
model2.to(device)
# Initialize optimizer
optimizer = optim.SGD(model2.parameters(), lr=0.001)
# Define the loss function with the class weights
criterion = nn.BCELoss()  # Binary classification loss
# Set the number of epochs
epochs = 20
```

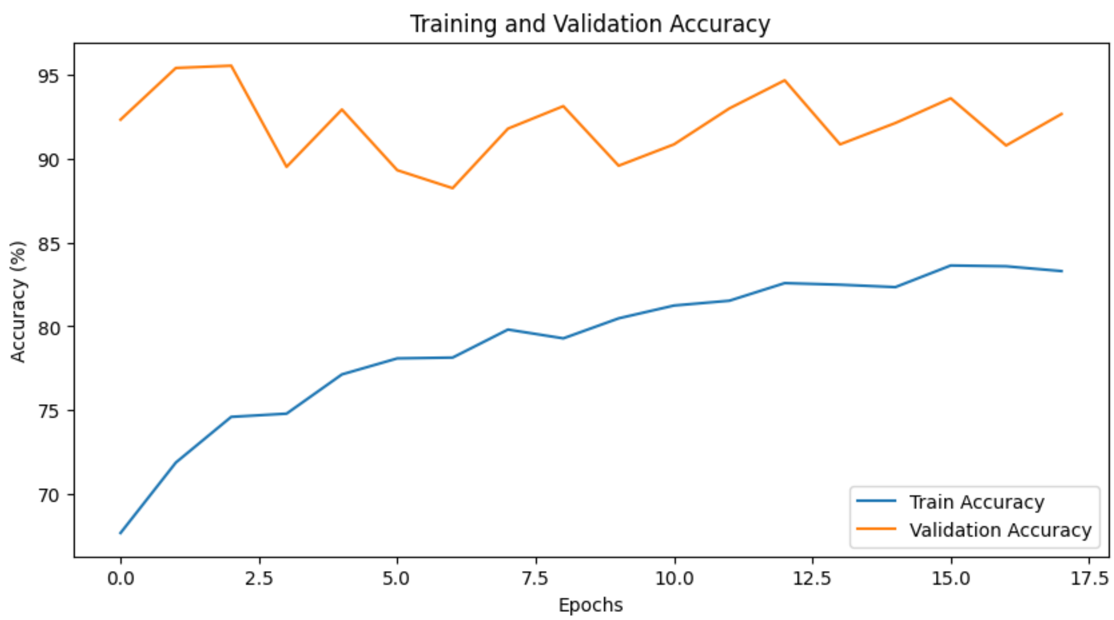
[11]:
```
train_and_validate(model2,train_dataloader, val_dataloader, criterion,␣
 ↪optimizer, epochs, device )
```

```
Epoch 1/20

Training Epoch 1: 100%|      | 33/33 [01:18<00:00,  2.39s/it,
train_loss=0.615]
Validating Epoch 1: 100%|      | 24/24 [00:25<00:00,  1.08s/it,
val_loss=0.619]

Epoch [1/20], Train Loss: 0.6151, Val Loss: 0.6193, Val Accuracy: 92.35%, Val
AUROC: 0.6333, Partial AUROC: 0.0372
Epoch 2/20

Training Epoch 2: 100%|      | 33/33 [01:15<00:00,  2.28s/it,
train_loss=0.569]
Validating Epoch 2: 100%|      | 24/24 [00:25<00:00,  1.05s/it,
val_loss=0.427]
```

```
Epoch [2/20], Train Loss: 0.5693, Val Loss: 0.4272, Val Accuracy: 95.44%, Val
AUROC: 0.7560, Partial AUROC: 0.0553
Epoch 3/20

Training Epoch 3: 100%|     | 33/33 [01:18<00:00,  2.37s/it,
train_loss=0.541]
Validating Epoch 3: 100%|     | 24/24 [00:25<00:00,  1.05s/it,
val_loss=0.336]

Epoch [3/20], Train Loss: 0.5412, Val Loss: 0.3361, Val Accuracy: 95.57%, Val
AUROC: 0.7691, Partial AUROC: 0.0580
Epoch 4/20

Training Epoch 4: 100%|     | 33/33 [01:14<00:00,  2.26s/it,
train_loss=0.522]
Validating Epoch 4: 100%|     | 24/24 [00:25<00:00,  1.07s/it,
val_loss=0.404]

Epoch [4/20], Train Loss: 0.5217, Val Loss: 0.4044, Val Accuracy: 89.53%, Val
AUROC: 0.7813, Partial AUROC: 0.0611
Epoch 5/20

Training Epoch 5: 100%|     | 33/33 [01:17<00:00,  2.34s/it,
train_loss=0.5]
Validating Epoch 5: 100%|     | 24/24 [00:25<00:00,  1.08s/it,
val_loss=0.325]

Epoch [5/20], Train Loss: 0.5001, Val Loss: 0.3252, Val Accuracy: 92.95%, Val
AUROC: 0.7870, Partial AUROC: 0.0620
Epoch 6/20

Training Epoch 6: 100%|     | 33/33 [01:16<00:00,  2.31s/it,
train_loss=0.491]
Validating Epoch 6: 100%|     | 24/24 [00:25<00:00,  1.07s/it,
val_loss=0.382]

Epoch [6/20], Train Loss: 0.4913, Val Loss: 0.3820, Val Accuracy: 89.33%, Val
AUROC: 0.8013, Partial AUROC: 0.0698
Epoch 7/20

Training Epoch 7: 100%|     | 33/33 [01:17<00:00,  2.34s/it,
train_loss=0.471]
Validating Epoch 7: 100%|     | 24/24 [00:27<00:00,  1.16s/it,
val_loss=0.377]

Epoch [7/20], Train Loss: 0.4713, Val Loss: 0.3771, Val Accuracy: 88.26%, Val
AUROC: 0.8055, Partial AUROC: 0.0716
Epoch 8/20

Training Epoch 8: 100%|     | 33/33 [01:23<00:00,  2.52s/it,
train_loss=0.456]
Validating Epoch 8: 100%|     | 24/24 [00:24<00:00,  1.03s/it,
val_loss=0.298]
```

```
Epoch [8/20], Train Loss: 0.4565, Val Loss: 0.2976, Val Accuracy: 91.81%, Val
AUROC: 0.8065, Partial AUROC: 0.0696
Epoch 9/20

Training Epoch 9: 100%|      | 33/33 [01:18<00:00,  2.37s/it,
train_loss=0.452]
Validating Epoch 9: 100%|      | 24/24 [00:23<00:00,  1.01it/s,
val_loss=0.27]

Epoch [9/20], Train Loss: 0.4515, Val Loss: 0.2701, Val Accuracy: 93.15%, Val
AUROC: 0.8119, Partial AUROC: 0.0729
Epoch 10/20

Training Epoch 10: 100%|      | 33/33 [01:14<00:00,  2.25s/it,
train_loss=0.44]
Validating Epoch 10: 100%|      | 24/24 [00:24<00:00,  1.03s/it,
val_loss=0.339]

Epoch [10/20], Train Loss: 0.4395, Val Loss: 0.3387, Val Accuracy: 89.60%, Val
AUROC: 0.8151, Partial AUROC: 0.0739
Epoch 11/20

Training Epoch 11: 100%|      | 33/33 [01:17<00:00,  2.35s/it,
train_loss=0.434]
Validating Epoch 11: 100%|      | 24/24 [00:26<00:00,  1.12s/it,
val_loss=0.301]

Epoch [11/20], Train Loss: 0.4335, Val Loss: 0.3011, Val Accuracy: 90.87%, Val
AUROC: 0.8142, Partial AUROC: 0.0716
Epoch 12/20

Training Epoch 12: 100%|      | 33/33 [01:16<00:00,  2.31s/it,
train_loss=0.422]
Validating Epoch 12: 100%|      | 24/24 [00:26<00:00,  1.12s/it,
val_loss=0.268]

Epoch [12/20], Train Loss: 0.4217, Val Loss: 0.2682, Val Accuracy: 93.02%, Val
AUROC: 0.8211, Partial AUROC: 0.0758
Epoch 13/20

Training Epoch 13: 100%|      | 33/33 [01:15<00:00,  2.29s/it,
train_loss=0.41]
Validating Epoch 13: 100%|      | 24/24 [00:24<00:00,  1.02s/it,
val_loss=0.225]

Epoch [13/20], Train Loss: 0.4102, Val Loss: 0.2249, Val Accuracy: 94.70%, Val
AUROC: 0.8217, Partial AUROC: 0.0751
Epoch 14/20

Training Epoch 14: 100%|      | 33/33 [01:15<00:00,  2.29s/it,
train_loss=0.407]
Validating Epoch 14: 100%|      | 24/24 [00:22<00:00,  1.08it/s,
val_loss=0.301]
```

```
Epoch [14/20], Train Loss: 0.4071, Val Loss: 0.3014, Val Accuracy: 90.87%, Val
AUROC: 0.8245, Partial AUROC: 0.0768
Epoch 15/20

Training Epoch 15: 100%|     | 33/33 [01:16<00:00,  2.31s/it,
train_loss=0.399]
Validating Epoch 15: 100%|     | 24/24 [00:19<00:00,  1.23it/s,
val_loss=0.274]

Epoch [15/20], Train Loss: 0.3994, Val Loss: 0.2740, Val Accuracy: 92.15%, Val
AUROC: 0.8313, Partial AUROC: 0.0818
Epoch 16/20

Training Epoch 16: 100%|     | 33/33 [01:12<00:00,  2.19s/it,
train_loss=0.394]
Validating Epoch 16: 100%|     | 24/24 [00:18<00:00,  1.27it/s,
val_loss=0.253]

Epoch [16/20], Train Loss: 0.3945, Val Loss: 0.2530, Val Accuracy: 93.62%, Val
AUROC: 0.8298, Partial AUROC: 0.0800
Epoch 17/20

Training Epoch 17: 100%|     | 33/33 [01:12<00:00,  2.19s/it,
train_loss=0.385]
Validating Epoch 17: 100%|     | 24/24 [00:17<00:00,  1.35it/s,
val_loss=0.292]

Epoch [17/20], Train Loss: 0.3845, Val Loss: 0.2920, Val Accuracy: 90.81%, Val
AUROC: 0.8347, Partial AUROC: 0.0837
Epoch 18/20

Training Epoch 18: 100%|     | 33/33 [01:19<00:00,  2.41s/it,
train_loss=0.386]
Validating Epoch 18: 100%|     | 24/24 [00:27<00:00,  1.13s/it,
val_loss=0.259]

Epoch [18/20], Train Loss: 0.3856, Val Loss: 0.2585, Val Accuracy: 92.68%, Val
AUROC: 0.8346, Partial AUROC: 0.0821
Early stopping triggered at epoch 18
Best Epoch: 13, Best Validation Loss: 0.2249
Training Complete
```



Classification Report:

	precision	recall	f1-score	support
Class 0	0.98	0.94	0.96	1431
Class 1	0.28	0.56	0.38	59
accuracy			0.93	1490

```
   macro avg       0.63      0.75      0.67      1490
weighted avg       0.95      0.93      0.94      1490
```

[11]:
```
CustomImageFeatureCNN2(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (fc_image): Linear(in_features=32768, out_features=512, bias=True)
  (fc_metadata): Linear(in_features=9, out_features=128, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc_combined): Linear(in_features=640, out_features=1, bias=True)
)
```

0.7 Model 3

[9]:
```
model3 = CustomImageFeatureCNN2(feature_input_size=9)  # Assuming 9 features␣
 ↪for metadata
model3.to(device)
# Initialize optimizer
optimizer = optim.SGD(model3.parameters(), lr=0.0001,weight_decay=1e-4)
# Define the loss function with the class weights
criterion = nn.BCELoss()  # Binary classification loss
# Set the number of epochs
epochs = 20
batch_size = 32
```

[10]:
```
train_dataloader = DataLoader(train_dataset, batch_size=batch_size,␣
 ↪shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=True)
```

[11]:
```
train_and_validate(model3,train_dataloader, val_dataloader, criterion,␣
 ↪optimizer, epochs, device )
```

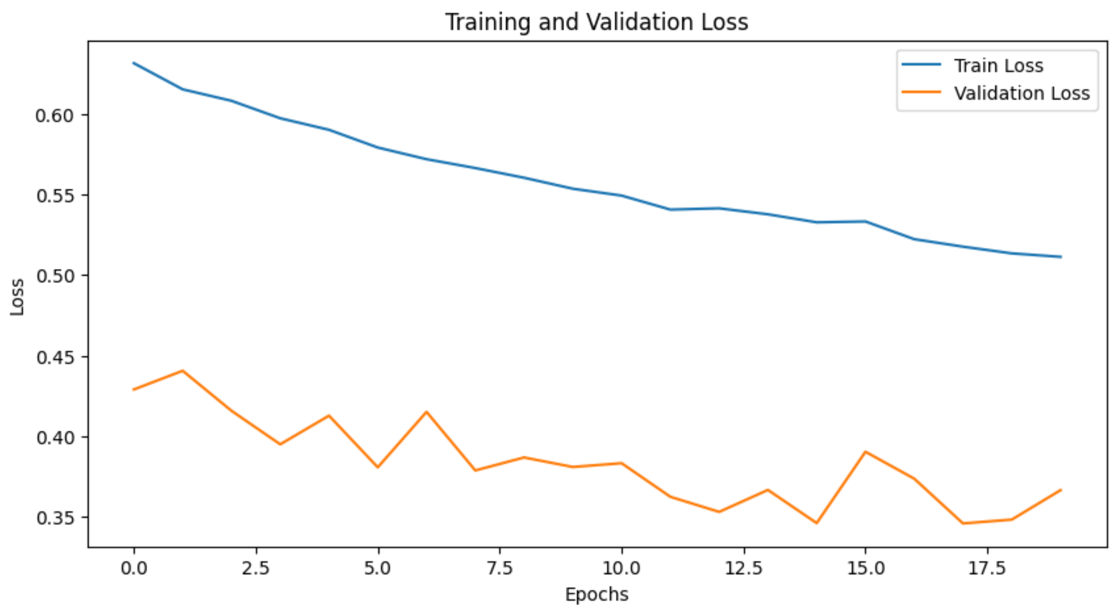
```
Epoch 1/20

Training Epoch 1: 100%|      | 66/66 [01:09<00:00,  1.05s/it,
train_loss=0.631]
Validating Epoch 1: 100%|      | 47/47 [00:20<00:00,  2.34it/s,
val_loss=0.429]
```

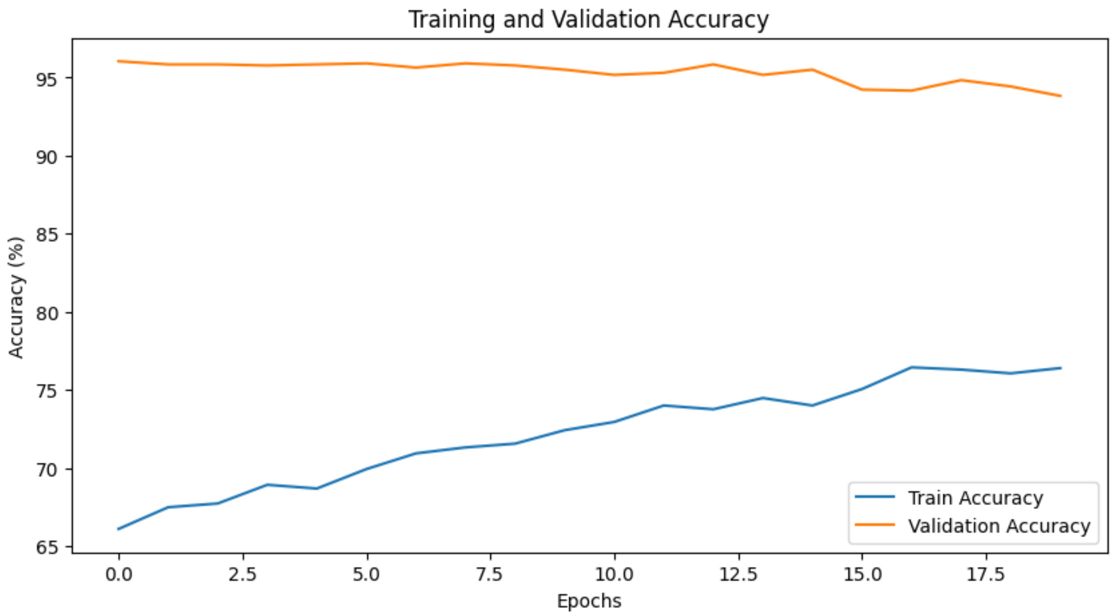
```
Epoch [1/20], Train Loss: 0.6314, Val Loss: 0.4291, Val Accuracy: 96.04%, Val
AUROC: 0.5977, Partial AUROC: 0.0416
Epoch 2/20

Training Epoch 2: 100%|     | 66/66 [01:10<00:00,  1.08s/it,
train_loss=0.615]
Validating Epoch 2: 100%|     | 47/47 [00:18<00:00,  2.60it/s,
val_loss=0.441]

Epoch [2/20], Train Loss: 0.6152, Val Loss: 0.4406, Val Accuracy: 95.84%, Val
AUROC: 0.6546, Partial AUROC: 0.0463
Epoch 3/20

Training Epoch 3: 100%|     | 66/66 [01:12<00:00,  1.09s/it,
train_loss=0.608]
Validating Epoch 3: 100%|     | 47/47 [00:17<00:00,  2.66it/s,
val_loss=0.416]

Epoch [3/20], Train Loss: 0.6081, Val Loss: 0.4158, Val Accuracy: 95.84%, Val
AUROC: 0.6802, Partial AUROC: 0.0496
Epoch 4/20

Training Epoch 4: 100%|     | 66/66 [01:11<00:00,  1.08s/it,
train_loss=0.597]
Validating Epoch 4: 100%|     | 47/47 [00:18<00:00,  2.54it/s,
val_loss=0.395]

Epoch [4/20], Train Loss: 0.5972, Val Loss: 0.3950, Val Accuracy: 95.77%, Val
AUROC: 0.6962, Partial AUROC: 0.0520
Epoch 5/20

Training Epoch 5: 100%|     | 66/66 [01:13<00:00,  1.12s/it,
train_loss=0.59]
Validating Epoch 5: 100%|     | 47/47 [00:18<00:00,  2.58it/s,
val_loss=0.413]

Epoch [5/20], Train Loss: 0.5901, Val Loss: 0.4128, Val Accuracy: 95.84%, Val
AUROC: 0.7199, Partial AUROC: 0.0534
Epoch 6/20

Training Epoch 6: 100%|     | 66/66 [01:08<00:00,  1.04s/it,
train_loss=0.579]
Validating Epoch 6: 100%|     | 47/47 [00:17<00:00,  2.64it/s,
val_loss=0.381]

Epoch [6/20], Train Loss: 0.5791, Val Loss: 0.3808, Val Accuracy: 95.91%, Val
AUROC: 0.7299, Partial AUROC: 0.0544
Epoch 7/20

Training Epoch 7: 100%|     | 66/66 [01:11<00:00,  1.08s/it,
train_loss=0.572]
Validating Epoch 7: 100%|     | 47/47 [00:18<00:00,  2.60it/s,
val_loss=0.415]
```

```
Epoch [7/20], Train Loss: 0.5719, Val Loss: 0.4152, Val Accuracy: 95.64%, Val
AUROC: 0.7443, Partial AUROC: 0.0548
Epoch 8/20

Training Epoch 8: 100%|     | 66/66 [01:11<00:00,  1.08s/it,
train_loss=0.566]
Validating Epoch 8: 100%|     | 47/47 [00:24<00:00,  1.94it/s,
val_loss=0.379]

Epoch [8/20], Train Loss: 0.5664, Val Loss: 0.3788, Val Accuracy: 95.91%, Val
AUROC: 0.7505, Partial AUROC: 0.0555
Epoch 9/20

Training Epoch 9: 100%|     | 66/66 [01:11<00:00,  1.09s/it,
train_loss=0.56]
Validating Epoch 9: 100%|     | 47/47 [00:17<00:00,  2.75it/s,
val_loss=0.387]

Epoch [9/20], Train Loss: 0.5603, Val Loss: 0.3869, Val Accuracy: 95.77%, Val
AUROC: 0.7552, Partial AUROC: 0.0564
Epoch 10/20

Training Epoch 10: 100%|     | 66/66 [01:08<00:00,  1.04s/it,
train_loss=0.554]
Validating Epoch 10: 100%|     | 47/47 [00:24<00:00,  1.89it/s,
val_loss=0.381]

Epoch [10/20], Train Loss: 0.5535, Val Loss: 0.3810, Val Accuracy: 95.50%, Val
AUROC: 0.7604, Partial AUROC: 0.0574
Epoch 11/20

Training Epoch 11: 100%|     | 66/66 [01:10<00:00,  1.06s/it,
train_loss=0.549]
Validating Epoch 11: 100%|     | 47/47 [00:17<00:00,  2.64it/s,
val_loss=0.383]

Epoch [11/20], Train Loss: 0.5493, Val Loss: 0.3833, Val Accuracy: 95.17%, Val
AUROC: 0.7665, Partial AUROC: 0.0574
Epoch 12/20

Training Epoch 12: 100%|     | 66/66 [01:14<00:00,  1.13s/it,
train_loss=0.541]
Validating Epoch 12: 100%|     | 47/47 [00:18<00:00,  2.51it/s,
val_loss=0.362]

Epoch [12/20], Train Loss: 0.5406, Val Loss: 0.3625, Val Accuracy: 95.30%, Val
AUROC: 0.7689, Partial AUROC: 0.0579
Epoch 13/20

Training Epoch 13: 100%|     | 66/66 [01:10<00:00,  1.07s/it,
train_loss=0.541]
Validating Epoch 13: 100%|     | 47/47 [00:18<00:00,  2.59it/s,
val_loss=0.353]
```

```
Epoch [13/20], Train Loss: 0.5414, Val Loss: 0.3531, Val Accuracy: 95.84%, Val
AUROC: 0.7737, Partial AUROC: 0.0586
Epoch 14/20

Training Epoch 14: 100%|     | 66/66 [01:10<00:00,  1.07s/it,
train_loss=0.538]
Validating Epoch 14: 100%|     | 47/47 [00:18<00:00,  2.56it/s,
val_loss=0.367]

Epoch [14/20], Train Loss: 0.5377, Val Loss: 0.3667, Val Accuracy: 95.17%, Val
AUROC: 0.7777, Partial AUROC: 0.0588
Epoch 15/20

Training Epoch 15: 100%|     | 66/66 [01:10<00:00,  1.07s/it,
train_loss=0.533]
Validating Epoch 15: 100%|     | 47/47 [00:18<00:00,  2.60it/s,
val_loss=0.346]

Epoch [15/20], Train Loss: 0.5327, Val Loss: 0.3462, Val Accuracy: 95.50%, Val
AUROC: 0.7787, Partial AUROC: 0.0586
Epoch 16/20

Training Epoch 16: 100%|     | 66/66 [01:10<00:00,  1.07s/it,
train_loss=0.533]
Validating Epoch 16: 100%|     | 47/47 [00:17<00:00,  2.62it/s,
val_loss=0.39]

Epoch [16/20], Train Loss: 0.5332, Val Loss: 0.3904, Val Accuracy: 94.23%, Val
AUROC: 0.7843, Partial AUROC: 0.0600
Epoch 17/20

Training Epoch 17: 100%|     | 66/66 [01:11<00:00,  1.09s/it,
train_loss=0.522]
Validating Epoch 17: 100%|     | 47/47 [00:18<00:00,  2.60it/s,
val_loss=0.374]

Epoch [17/20], Train Loss: 0.5222, Val Loss: 0.3737, Val Accuracy: 94.16%, Val
AUROC: 0.7845, Partial AUROC: 0.0596
Epoch 18/20

Training Epoch 18: 100%|     | 66/66 [01:11<00:00,  1.09s/it,
train_loss=0.518]
Validating Epoch 18: 100%|     | 47/47 [00:20<00:00,  2.28it/s,
val_loss=0.346]

Epoch [18/20], Train Loss: 0.5176, Val Loss: 0.3460, Val Accuracy: 94.83%, Val
AUROC: 0.7861, Partial AUROC: 0.0597
Epoch 19/20

Training Epoch 19: 100%|     | 66/66 [01:10<00:00,  1.07s/it,
train_loss=0.513]
Validating Epoch 19: 100%|     | 47/47 [00:17<00:00,  2.62it/s,
val_loss=0.348]
```

```
Epoch [19/20], Train Loss: 0.5134, Val Loss: 0.3483, Val Accuracy: 94.43%, Val
AUROC: 0.7861, Partial AUROC: 0.0602
Epoch 20/20

Training Epoch 20: 100%|      | 66/66 [01:12<00:00,  1.10s/it,
train_loss=0.511]
Validating Epoch 20: 100%|      | 47/47 [00:19<00:00,  2.47it/s,
val_loss=0.367]

Epoch [20/20], Train Loss: 0.5113, Val Loss: 0.3666, Val Accuracy: 93.83%, Val
AUROC: 0.7901, Partial AUROC: 0.0610
Best Epoch: 18, Best Validation Loss: 0.3460
Training Complete
```






```
Classification Report:
              precision    recall  f1-score   support

     Class 0       0.97      0.96      0.97      1431
     Class 1       0.29      0.37      0.32        59

    accuracy                           0.94      1490
   macro avg       0.63      0.67      0.65      1490
weighted avg       0.95      0.94      0.94      1490
```

[11]:
```
CustomImageFeatureCNN2(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (fc_image): Linear(in_features=32768, out_features=512, bias=True)
  (fc_metadata): Linear(in_features=9, out_features=128, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc_combined): Linear(in_features=640, out_features=1, bias=True)
```

```
)
```

[ ]:

```

```