Putting_it_all_together

December 9, 2024

[1]:
```python
# Standard Libraries
import io
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 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
from torchvision import models
# Image Processing
from PIL import Image
from torchvision import transforms, models
import cv2

# 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

#sklearn
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split

#Visualization
```

```
import plotly.express as px
import plotly.graph_objects as go
```

1 1) Problem Statement

Skin cancer is the most common form of cancer in the United States and ranks 17th globally (WCRF).There are three major types of skin cancer—Basal Cell Carcinoma (BCC), Squamous Cell Carcinoma (SCC), and Melanoma. While BCC and SCC are considered less lethal, melanoma is the deadliest form ofskin cancer. It is expected to be diagnosed over 200,000 times in the US in 2024, with nearly 9,000 deathsprojected. Automated image analysis tools play a significant role in expediting clinical presentation anddiagnosis, positively impacting hundreds of thousands of people each year.For a telehealth app company, addressing the challenge of skin cancer detection in underserved populations or non-clinical settings is particularly significant. Current diagnostic methods rely on high-quality dermatoscope images, which are typically captured in dermatology clinics. These images reveal morphological features not visible to the naked eye. To provide this early detection service on our platform, we need to develop an algorithm capable of accurately classifying lower-quality malignant skin lesions from benign ones. Additionally, this algorithm should assist in diagnosing users based on their type of lesions and personal information.

2 2) Data Ingestion

From the Data Ingestion to Data Preprocessing stage, I utilized the original dataset prior to resampling, which is too large to upload to GitHub. As a result, you may encounter an error stating "No such file exists." To address this limitation, I discussed the issue of data size constraints with the professor. Subsequently, after resampling the dataset, I proceeded with data preprocessing and used the resampled data for the subsequent stages, including Feature Engineering and Model Development. This approach allowed me to handle the imbalanced dataset effectively while aligning with the constraints of data storage and accessibility.

2.1 Load Data

[3]:
```
data = pd.read_csv('../data/raw/train-metadata.csv')
```

```
/tmp/ipykernel_3449613/2095587616.py:1: DtypeWarning: Columns (51,52) have mixed
types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv('../data/raw/train-metadata.csv')
```

3 3) Explolatory Data Analysis

3.1 Missing Value Analysis

[4]:
```
def df_stats(df: pd.DataFrame, include_all: bool = False):
    """
    Print statistics and null value counts for a pandas DataFrame.

    Parameters:
        df (pd.DataFrame): The DataFrame to analyze.
        include_all (bool): If True, include all columns in the descriptive␣
↪statistics; otherwise, include only numeric columns.

    Returns:
        None
    """
    if df.empty:
        print("The DataFrame is empty.")
        return

    # Print descriptive statistics
    print("Descriptive Statistics:")
    if include_all:
        print(df.describe(include='all'))
    else:
        print(df.describe(include=[np.number]))
    print("\n" + "-"*50 + "\n")  # Separator for clarity

    # Print the number of null values per column
    print("Null Value Counts:")
    print(df.isnull().sum())
    print("\n" + "-"*50 + "\n")  # Separator for clarity

    # Additional information: Percentage of null values per column
    print("Percentage of Null Values:")
    print(df.isnull().mean() * 100)
    print("\n" + "-"*50 + "\n")  # Separator for clarity

    # Number of rows and columns
    print(f"Number of rows: {df.shape[0]}")
    print(f"Number of columns: {df.shape[1]}")
    print("\n" + "-"*50 + "\n")  # Separator for clarity
```

[5]:
```
df_stats(data)
```

```
Descriptive Statistics:
                target     age_approx  clin_size_long_diam_mm        tbp_lv_A  \
count   401059.000000   398261.000000           401059.000000   401059.000000
```

mean	0.000980	58.012986	3.930827	19.974007
std	0.031288	13.596165	1.743068	3.999489
min	0.000000	5.000000	1.000000	-2.487115
25%	0.000000	50.000000	2.840000	17.330821
50%	0.000000	60.000000	3.370000	19.801910
75%	0.000000	70.000000	4.380000	22.304628
max	1.000000	85.000000	28.400000	48.189610

	tbp_lv_Aext	tbp_lv_B	tbp_lv_Bext	tbp_lv_C	\
count	401059.000000	401059.000000	401059.000000	401059.000000	
mean	14.919247	28.281706	26.913015	34.786341	
std	3.529384	5.278676	4.482994	5.708469	
min	-9.080269	-0.730989	9.237066	3.054228	
25%	12.469740	24.704372	23.848125	31.003148	
50%	14.713930	28.171570	26.701704	34.822580	
75%	17.137175	31.637429	29.679913	38.430298	
max	37.021680	54.306900	48.372700	58.765170	

	tbp_lv_Cext	tbp_lv_H	…	tbp_lv_radial_color_std_max	\
count	401059.000000	401059.000000	…	401059.000000	
mean	30.921279	54.653689	…	1.016459	
std	4.829345	5.520849	…	0.734631	
min	11.846520	-1.574164	…	0.000000	
25%	27.658285	51.566273	…	0.563891	
50%	30.804893	55.035632	…	0.902281	
75%	33.963868	58.298184	…	1.334523	
max	54.305290	105.875784	…	11.491140	

	tbp_lv_stdL	tbp_lv_stdLExt	tbp_lv_symm_2axis	\
count	401059.000000	401059.000000	401059.000000	
mean	2.715190	2.238605	0.306823	
std	1.738165	0.623884	0.125038	
min	0.268160	0.636247	0.052034	
25%	1.456570	1.834745	0.211429	
50%	2.186693	2.149758	0.282297	
75%	3.474565	2.531443	0.382022	
max	17.563650	25.534791	0.977055	

	tbp_lv_symm_2axis_angle	tbp_lv_x	tbp_lv_y	tbp_lv_z	\
count	401059.000000	401059.000000	401059.000000	401059.000000	
mean	86.332073	-3.091862	1039.598221	55.823389	
std	52.559511	197.257995	409.819653	87.968245	
min	0.000000	-624.870728	-1052.134000	-291.890442	
25%	40.000000	-147.022125	746.519673	-8.962647	
50%	90.000000	-5.747253	1172.803000	67.957947	
75%	130.000000	140.474835	1342.131540	126.611567	
max	175.000000	614.471700	1887.766846	319.407000	

```
       mel_thick_mm  tbp_lv_dnn_lesion_confidence
count     63.000000                  4.010590e+05
mean       0.670952                  9.716220e+01
std        0.792798                  8.995782e+00
min        0.200000                  1.261082e-16
25%        0.300000                  9.966882e+01
50%        0.400000                  9.999459e+01
75%        0.600000                  9.999996e+01
max        5.000000                  1.000000e+02

[8 rows x 37 columns]

--------------------------------------------------

Null Value Counts:
isic_id                              0
target                               0
patient_id                           0
age_approx                        2798
sex                              11517
anatom_site_general               5756
clin_size_long_diam_mm               0
image_type                           0
tbp_tile_type                        0
tbp_lv_A                             0
tbp_lv_Aext                          0
tbp_lv_B                             0
tbp_lv_Bext                          0
tbp_lv_C                             0
tbp_lv_Cext                          0
tbp_lv_H                             0
tbp_lv_Hext                          0
tbp_lv_L                             0
tbp_lv_Lext                          0
tbp_lv_areaMM2                       0
tbp_lv_area_perim_ratio              0
tbp_lv_color_std_mean                0
tbp_lv_deltaA                        0
tbp_lv_deltaB                        0
tbp_lv_deltaL                        0
tbp_lv_deltaLB                       0
tbp_lv_deltaLBnorm                   0
tbp_lv_eccentricity                  0
tbp_lv_location                      0
tbp_lv_location_simple               0
tbp_lv_minorAxisMM                   0
tbp_lv_nevi_confidence               0
tbp_lv_norm_border                   0
```

```
tbp_lv_norm_color                 0
tbp_lv_perimeterMM                0
tbp_lv_radial_color_std_max       0
tbp_lv_stdL                       0
tbp_lv_stdLExt                    0
tbp_lv_symm_2axis                 0
tbp_lv_symm_2axis_angle           0
tbp_lv_x                          0
tbp_lv_y                          0
tbp_lv_z                          0
attribution                       0
copyright_license                 0
lesion_id                    379001
iddx_full                         0
iddx_1                            0
iddx_2                       399991
iddx_3                       399994
iddx_4                       400508
iddx_5                       401058
mel_mitotic_index            401006
mel_thick_mm                 400996
tbp_lv_dnn_lesion_confidence      0
dtype: int64

--------------------------------------------------

Percentage of Null Values:
isic_id                        0.000000
target                         0.000000
patient_id                     0.000000
age_approx                     0.697653
sex                            2.871647
anatom_site_general            1.435200
clin_size_long_diam_mm         0.000000
image_type                     0.000000
tbp_tile_type                  0.000000
tbp_lv_A                       0.000000
tbp_lv_Aext                    0.000000
tbp_lv_B                       0.000000
tbp_lv_Bext                    0.000000
tbp_lv_C                       0.000000
tbp_lv_Cext                    0.000000
tbp_lv_H                       0.000000
tbp_lv_Hext                    0.000000
tbp_lv_L                       0.000000
tbp_lv_Lext                    0.000000
tbp_lv_areaMM2                 0.000000
tbp_lv_area_perim_ratio        0.000000
```

```
tbp_lv_color_std_mean           0.000000
tbp_lv_deltaA                   0.000000
tbp_lv_deltaB                   0.000000
tbp_lv_deltaL                   0.000000
tbp_lv_deltaLB                  0.000000
tbp_lv_deltaLBnorm              0.000000
tbp_lv_eccentricity             0.000000
tbp_lv_location                 0.000000
tbp_lv_location_simple          0.000000
tbp_lv_minorAxisMM              0.000000
tbp_lv_nevi_confidence          0.000000
tbp_lv_norm_border              0.000000
tbp_lv_norm_color               0.000000
tbp_lv_perimeterMM              0.000000
tbp_lv_radial_color_std_max     0.000000
tbp_lv_stdL                     0.000000
tbp_lv_stdLExt                  0.000000
tbp_lv_symm_2axis               0.000000
tbp_lv_symm_2axis_angle         0.000000
tbp_lv_x                        0.000000
tbp_lv_y                        0.000000
tbp_lv_z                        0.000000
attribution                     0.000000
copyright_license               0.000000
lesion_id                      94.500061
iddx_full                       0.000000
iddx_1                          0.000000
iddx_2                         99.733705
iddx_3                         99.734453
iddx_4                         99.862614
iddx_5                         99.999751
mel_mitotic_index              99.986785
mel_thick_mm                   99.984292
tbp_lv_dnn_lesion_confidence    0.000000
dtype: float64

--------------------------------------------------

Number of rows: 401059
Number of columns: 55

--------------------------------------------------
```

In the application I am developing, users will input an image and provide personal information. To ensure transparency and a user-centric design, only metadata accessible to users will be used as predictors in the model. Consequently, I have selected the fol-

lowing metadata fields for inclusion: "age_approx", "sex", "anatom_site_general", and "clin_size_long_diam_mm". These fields are both relevant to the prediction task and available to users.

From the data analysis, "age_approx", "sex", and "anatom_site_general" were identified as having missing values. However, the percentage of missing data for these fields is manageable, allowing for imputation strategies such as using the median for numerical fields like "age_approx" and the mode for categorical fields like "sex" and "anatom_site_general." This ensures the completeness and reliability of the metadata while maintaining the model's predictive performance.

3.2 Visualize Target Variable

[6]:
```
# Target Distribution

# Count the occurrences of each target value and sort by index
target_counts = data['target'].value_counts().sort_index()

# Calculate the total number of samples in the training DataFrame
total = len(data)

# Create a list of percentages for each target class, formatted as a string
percentage = [f'{count/total:0.3%}' for count in target_counts]

# Create a bar plot to visualize the distribution of the target variable
fig = go.Figure(data=[
    go.Bar(
        x=target_counts.index,  # X-axis represents the unique target classes
        y=target_counts.values,  # Y-axis represents the counts of each class
        text=percentage,  # Display percentages on top of the bars
        textposition='auto'  # Automatically position text on bars
    )
])

# Update layout of the plot with titles and formatting
fig.update_layout(
    title='Distribution of Target Variable',  # Main title of the plot
    xaxis_title='Lesion Class',  # Title for the X-axis
    yaxis_title='Count',  # Title for the Y-axis
    template='plotly_white',  # Use a white background for the plot
    height=600, width=1200  # Set the dimensions of the plot
)

# Set the y-axis to a logarithmic scale to better visualize class distributions
fig.update_layout(yaxis=dict(type='log'))

# Add an annotation to show the total number of samples in the dataset
```

```
fig.add_annotation(
    text=f"<b>TOTAL SAMPLES:  {total:,}</b>",  # Format total count with commas
    xref="paper", yref="paper",  # Reference the entire paper for positioning
    x=0.98, y=1.05,  # Position the annotation near the top-right corner
    showarrow=False,  # Do not show an arrow pointing to the text
    font=dict(size=12)  # Set the font size for the annotation
)

# Display the plot
fig.show()
```

From the target distribution graph, it is evident that the dataset is highly imbalanced. Class 1, representing malignant cases, constitutes only 0.098% of the total data, while Class 0, representing benign cases, accounts for 99.902%. This extreme imbalance poses challenges for the model, as it may struggle to adequately learn patterns for the minority class, potentially leading to biased predictions heavily favoring the majority class. Addressing this imbalance is crucial to ensure the model's effectiveness and fairness, particularly for detecting malignant cases.

3.3 Visualize categorical features

[7]:
```
def plot_categorical_feature_distribution(
    df: pd.DataFrame,
    feature_col: str,
    target_col: str = 'target',
    target_as_str: bool = True,
    log_y: bool = False,
    template_theme: str = "plotly_white",
    group_by_target: bool = True,
    stack_bar: bool = False
) -> None:
    """
    Plots the distribution of a categorical feature, optionally grouped by a␣
 ↪target variable.

    Args:
        df (pd.DataFrame): The DataFrame containing the data.
        feature_col (str): The name of the categorical feature column to plot.
        target_col (str, optional): The name of the target column. Defaults to␣
 ↪'target'.
        target_as_str (bool, optional): Whether to treat target variable as␣
 ↪strings. Defaults to True.
        log_y (bool, optional): Whether to use a logarithmic scale for the␣
 ↪Y-axis. Defaults to False.
        template_theme (str, optional): Plotly template theme to use. Defaults␣
 ↪to 'plotly_white'.
```

```
        group_by_target (bool, optional): Whether to group bars by target␣
↪variable. Defaults to True.
        stack_bar (bool, optional): Whether to stack bars instead of grouping.␣
↪Defaults to False.

    Returns:
        None; displays the plot.
    """

    # Create a copy of the DataFrame and sort it based on feature and target␣
↪columns
    _df = df.copy().sort_values(by=[feature_col, target_col]).
↪reset_index(drop=True)

    # Check if we need to group the bars by the target variable
    if group_by_target:
        # Create a histogram plot grouped by the target variable
        fig = px.histogram(
            _df, x=feature_col, color=target_col,
            log_y=log_y, height=500, width=1200, template=template_theme,
            title=f'Distribution of {feature_col.upper()} By TARGET',
            barmode='group' if not stack_bar else 'stack'  # Choose between␣
↪grouped or stacked bars
        )
    else:
        # Create a histogram plot without grouping by the target variable
        fig = px.histogram(
            _df, x=feature_col, color=feature_col,
            log_y=log_y, height=500, width=1200, template=template_theme,
            title=f'<b>DISTRIBUTION OF {feature_col.replace("_", " ").upper()}',
        )

    # Update the layout of the plot with titles and gaps
    fig.update_layout(
        bargap=0.1,  # Set the gap between bars
        xaxis_title=f"{feature_col.title()}",  # Format the X-axis title
        yaxis_title="Count",  # Title for the Y-axis
        showlegend=group_by_target  # Show legend only when grouped by target
    )

    # Apply log scale to the Y-axis if requested
    if log_y:
        fig.update_layout(yaxis_type="log")

    # Display the plot
    fig.show()
```

3.3.1 Age_approx

[8]:
```
plot_categorical_feature_distribution(data, "age_approx", group_by_target=False)
```

[9]:
```
plot_categorical_feature_distribution(data, "age_approx", group_by_target=True,␣
↪stack_bar=False, log_y=True)
```

3.3.2 Anatom_Site_General

[10]:
```
plot_categorical_feature_distribution(data, "anatom_site_general",␣
↪group_by_target=True, stack_bar=False, log_y = True)
```

3.3.3 Sex

[11]:
```
plot_categorical_feature_distribution(data, "sex", group_by_target=True,␣
↪stack_bar=False, log_y=True)
```

From these graphs, I found out that age groups under 40 and females, in particular, are underrepresented for malignant cases. This could lead to lower recall for these subgroups, as the model may not learn enough from the available data.

3.4 Visualize continuous features

[12]:
```
def plot_continuous_feature_distribution(
    df: pd.DataFrame,
    feature_col: str,
    plot_style: str = "histogram",
    feature_readable_name: str | None = None,
    target_col: str = "target",
    log_y: bool = False,
    template_theme: str = "plotly_white",
    group_by_target: bool = True,
    n_bins: int = 50
) -> None:
    """
    Plots the distribution of a continuous feature in the DataFrame.

    Args:
        df (pd.DataFrame): The DataFrame containing the feature and target␣
↪columns.
        feature_col (str): The name of the feature column to plot.
        plot_style (str, optional): The style of the plot ('histogram' or␣
↪'box'). Defaults to 'histogram'.
        feature_readable_name (str | None, optional): A readable name for the␣
↪feature to use in the title. Defaults to None.
```

```
        target_col (str, optional): The name of the target column. Defaults to␣
↪'target'.
        log_y (bool, optional): Whether to apply a logarithmic scale to the␣
↪y-axis. Defaults to False.
        template_theme (str, optional): The Plotly template theme to use for␣
↪the plot. Defaults to 'plotly_white'.
        group_by_target (bool, optional): Whether to group the plot by the␣
↪target variable. Defaults to True.
        n_bins (int, optional): The number of bins to use for the histogram.␣
↪Defaults to 50.

    Raises:
        TypeError: If df is not a pandas DataFrame.
        ValueError: If feature_col or target_col are not found in the DataFrame␣
↪or if plot_style is invalid.

    Returns:
        None: Displays the plot.
    """

    # Input validation
    if not isinstance(df, pd.DataFrame):
        raise TypeError("Input 'df' must be a pandas DataFrame.")
    if feature_col not in df.columns:
        raise ValueError(f"Feature column '{feature_col}' not found in␣
↪DataFrame.")
    if target_col not in df.columns:
        raise ValueError(f"Target column '{target_col}' not found in DataFrame.
↪")
    if plot_style not in ["histogram", "box"]:
        raise ValueError("Invalid plot_style. Choose either 'histogram' or␣
↪'box'.")

    # Make a copy of the DataFrame to avoid modifying the original data
    _df = df.copy().sort_values(by=[feature_col, target_col]).
↪reset_index(drop=True)

    # Plotting logic based on the chosen plot style
    if plot_style == "histogram":
        if group_by_target:
            # Create a histogram for each target value
            fig = go.Figure()
            for target_value in _df[target_col].unique():
                subset = _df[_df[target_col] == target_value]
                fig.add_trace(go.Histogram(
                    x=subset[feature_col],
```

```
                name=str(target_value),
                opacity=0.7,
                nbinsx=n_bins
            ))

        # Update layout for overlay histogram
        fig.update_layout(
            barmode='overlay',
            title=f"Distribution of {feature_readable_name or feature_col.
↪upper()} by Target",
            height=500, width=1200, template=template_theme,
            xaxis_title=feature_readable_name or feature_col,
            yaxis_title="Count",
            showlegend=True
        )
    else:
        # Create a single histogram without grouping
        fig = px.histogram(
            _df, x=feature_col, log_y=log_y, height=500, width=1200,␣
↪template=template_theme,
            title=f"Distribution of {feature_readable_name or feature_col.
↪upper()}",
            nbins=n_bins
        )

        # Update layout for single histogram
        fig.update_layout(
            xaxis_title=feature_readable_name or feature_col,
            yaxis_title="Count",
            showlegend=False
        )

elif plot_style == "box":
    if group_by_target:
        # Create a box plot for each target value
        fig = go.Figure()
        for target_value in _df[target_col].unique():
            subset = _df[_df[target_col] == target_value]
            fig.add_trace(go.Box(
                y=subset[feature_col],
                name=str(target_value),
                boxpoints='outliers',  # Show outliers
                boxmean=True  # Show mean in the box plot
            ))

        # Update layout for box plot grouped by target
        fig.update_layout(
```

```
                title=f'Distribution of {feature_readable_name or feature_col.
↪upper()} by Target (includes likely outliers)',
                height=500, width=1200, template=template_theme,
                xaxis_title='Target',
                yaxis_title=f'{feature_readable_name or feature_col}',
                showlegend=True
            )
        else:
            # Create a single box plot without grouping
            fig = px.box(
                _df, y=feature_col,
                height=500,
                width=1200,
                template=template_theme,
                title=f"Distribution of {feature_readable_name or feature_col.
↪upper()}",
                points="outliers",  # Show outliers
            )

            # Update layout for single box plot
            fig.update_layout(
                yaxis_title=f'{feature_readable_name or feature_col}',
                showlegend=False
            )

    # Apply log scale to y-axis if requested (only for histogram)
    if log_y and plot_style == "histogram":
        fig.update_layout(yaxis_type='log')

    # Display the plot
    fig.show()
```

3.4.1 clin_size_long_diam_mm

[13]:
```
plot_continuous_feature_distribution(data, 'clin_size_long_diam_mm',␣
↪plot_style="box", log_y=True, group_by_target=True)
```

The boxplot above shows significant outliers in the "clin_size_long_diam_mm" feature for both classes, especially Class 0. These outliers can negatively impact the training of a neural network by skewing the weight updates

[14]:
```
plot_continuous_feature_distribution(data, 'clin_size_long_diam_mm',␣
↪plot_style="histogram", log_y=True, group_by_target=True, n_bins=100)
```

3.5 Visualize Images

[15]:
```
#Load image from hdf5 file
def load_image_from_hdf5(isic_id: str,
                         file_path: str = "../data/raw/train-image.hdf5",
                         n_channels: int = 3):
    # Handle the case where the isic_id is passed incorrectly
    if not isic_id.lower().startswith("isic"):
        isic_id = f"ISIC_{int(str(isic_id).split('_', 1)[-1]):>07}"

    # Open the HDF5 file in read mode
    with h5py.File(file_path, 'r') as hf:

        # Retrieve the image data from the HDF5 dataset using the provided ISIC␣
↪ID
        try:
            image_data = hf[isic_id][()]
        except KeyError:
            raise KeyError(f"ISIC ID {isic_id} not found in HDF5 file.")

        # Convert the binary data to a numpy array
        image_array = np.frombuffer(image_data, np.uint8)

        # Decode the image from the numpy array
        if n_channels == 3:
            # Load the image as a color image (BGR) and convert to RGB
            image = cv2.cvtColor(cv2.imdecode(image_array, cv2.IMREAD_COLOR),␣
↪cv2.COLOR_BGR2RGB)
        else:
            # Load the image as a grayscale image
            image = cv2.imdecode(image_array, cv2.IMREAD_GRAYSCALE)

        # If the image failed to load for some reason (problems decoding) ...
        if image is None:
            raise ValueError(f"Could not decode image for ISIC ID: {isic_id}")

        return image
```

[23]:
```
def plot_images_by_target(df: pd.DataFrame, target_value: int, max_images: int␣
↪= 10) -> None:
    """Load and plot images based on the target value.

    Args:
        processed_df (pd.DataFrame): The DataFrame containing image metadata.
        target_value (int): The target value to filter images.
        max_images (int, optional): Maximum number of images to display.␣
↪Defaults to 10.
```

```
    Returns:
        None; displays a plot of the images.
    """
    # Validate inputs
    if not isinstance(target_value, int):
        raise ValueError("target_value must be an integer.")
    if not isinstance(max_images, int) or max_images <= 0:
        raise ValueError("max_images must be a positive integer.")

    # Filter the DataFrame for the specified target value and limit the number␣
↪of images
    filtered_df = df[df['target'] == target_value].head(max_images)

    images = []  # Initialize a list to hold the loaded images
    for isic_id in filtered_df['isic_id']:
        try:
            # Load the image using the provided ISIC ID from the HDF5 file
            image = load_image_from_hdf5(isic_id)
            images.append(image)  # Append the loaded image to the list
        except Exception as e:
            print(f"Error loading image for ISIC ID {isic_id}: {e}")

    # Create a DataFrame to store the loaded images along with their metadata
    image_df = pd.DataFrame({
        'isic_id': filtered_df['isic_id'],
        'target': filtered_df['target'],
        'image': images
    })

    n_images = len(image_df)  # Get the number of images to display
    fig, axes = plt.subplots(1, n_images, figsize=(15, 5))  # Create a subplot␣
↪for each image
    fig.suptitle(f'Images of Lesions with Target Value {target_value}',␣
↪fontsize=14)  # Main title

    # Iterate over the axes, ISIC IDs, and images to display each image
    for ax, isic_id, img in zip(axes, image_df['isic_id'], image_df['image']):
        ax.imshow(img)  # Display the image
        ax.set_title(f'ISIC ID: {isic_id}', fontsize=5)  # Set the title for␣
↪each image
        ax.axis('off')  # Hide the axis

    plt.tight_layout()  # Adjust layout to make room for the main title
    plt.show()  # Display the plot
```

[24]:
```
plot_images_by_target(data, target_value=1, max_images=10)
```


```
[25]: plot_images_by_target(data, target_value=0, max_images=10)
```

3.6 Correlation Analysis

To perform correlation analysis, I must first split the dataset into training, validation, and test sets. The training data is then used for the correlation analysis due to computational constraints that prevent me from using the entire dataset. By focusing on the training data, I ensure that the subset adequately represents the overall dataset while remaining manageable for analysis. This allows for meaningful computation of correlation coefficients and effective visualization of feature relationships using a heatmap.

3.6.1 Split Data inot Train, Validation and Test

I split the data into 70% train, 15% validation and 15% validation

```
[5]: ## Load data from the "train-metadata.csv file"  and split into train, val, test

try:
    data = pd.read_csv('../data/raw/train-metadata.csv')
except FileNotFoundError:
    print("Error: The specified CSV file was not found.")
    raise  # Re-raise the error after logging
except pd.errors.EmptyDataError:
    print("Error: The CSV file is empty.")
    raise
except pd.errors.ParserError:
    print("Error: The CSV file could not be parsed.")
```

```
    raise

# Select features (X) and the target variable (y)
try:
    X = data[['isic_id', 'age_approx', 'sex', 'anatom_site_general',␣
↪'clin_size_long_diam_mm']]
    y = data['target']
except KeyError as e:
    print(f"Error: Missing expected column in the dataset: {e}")
    raise

# Split the data into training and temporary sets (70% train, 30% temp)
try:
    X_train, X_temp, y_train, y_temp = train_test_split(
        X, y,
        test_size=0.3,
        random_state=88,
        stratify=y  # Ensures the target variable distribution is preserved
    )
except ValueError as e:
    print(f"Error during train-test split: {e}")
    raise

# Further split the temporary set into validation and test sets (15% val, 15%␣
↪test)
try:
    X_val, X_test, y_val, y_test = train_test_split(
        X_temp, y_temp,
        test_size=0.5,  # This effectively splits the 30% temp into two equal␣
↪parts
        random_state=88,
        stratify=y_temp  # Again preserves the target variable distribution
    )
except ValueError as e:
    print(f"Error during validation-test split: {e}")
    raise

# Create DataFrames for the training, validation, and test sets
train_df = pd.concat([X_train, y_train], axis=1)
validation_df = pd.concat([X_val, y_val], axis=1)
test_df = pd.concat([X_test, y_test], axis=1)

# Save the processed DataFrames to CSV files
try:
    train_df.to_csv('../data/processed/train-metadata.csv', index=False)
    validation_df.to_csv('../data/processed/validation-metadata.csv',␣
↪index=False)
```

```
    test_df.to_csv('../data/processed/test-metadata.csv', index=False)
except Exception as e:
    print(f"Error while saving CSV files: {e}")
    raise
```

```
/tmp/ipykernel_3501635/3409806486.py:4: DtypeWarning: Columns (51,52) have mixed
types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv('../data/raw/train-metadata.csv')
```

3.6.2 Preprocessing Pipeline

[6]:
```
# Custom transformer for handling missing values
class MissingValueHandler(BaseEstimator, TransformerMixin):
    # Fit method, not modifying any parameters, just returning self
    def fit(self, X, y=None):
        return self

    # Transform method to handle missing values
    def transform(self, X):
        # Ensure input is a pandas DataFrame
        if not isinstance(X, pd.DataFrame):
            raise TypeError("Input must be a pandas DataFrame.")

        # Identify numerical columns
        num_cols = X.select_dtypes(include=['int64', 'float64']).columns
        # Identify categorical columns
        cat_cols = X.select_dtypes(include=['object', 'category']).columns

        # Create imputer for numerical data using median
        num_imputer = SimpleImputer(strategy="median")
        # Apply imputer to numerical columns
        X[num_cols] = num_imputer.fit_transform(X[num_cols])

        # Create imputer for categorical data using the most frequent value
        cat_imputer = SimpleImputer(strategy="most_frequent")
        # Apply imputer to categorical columns
        X[cat_cols] = cat_imputer.fit_transform(X[cat_cols])

        return X  # Return the transformed DataFrame

# Custom transformer for one-hot encoding
class OneHotEncoderTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        # Initialize the OneHotEncoder with specified parameters
        self.encoder = OneHotEncoder(sparse_output=False,␣
 ↪handle_unknown="ignore")
```

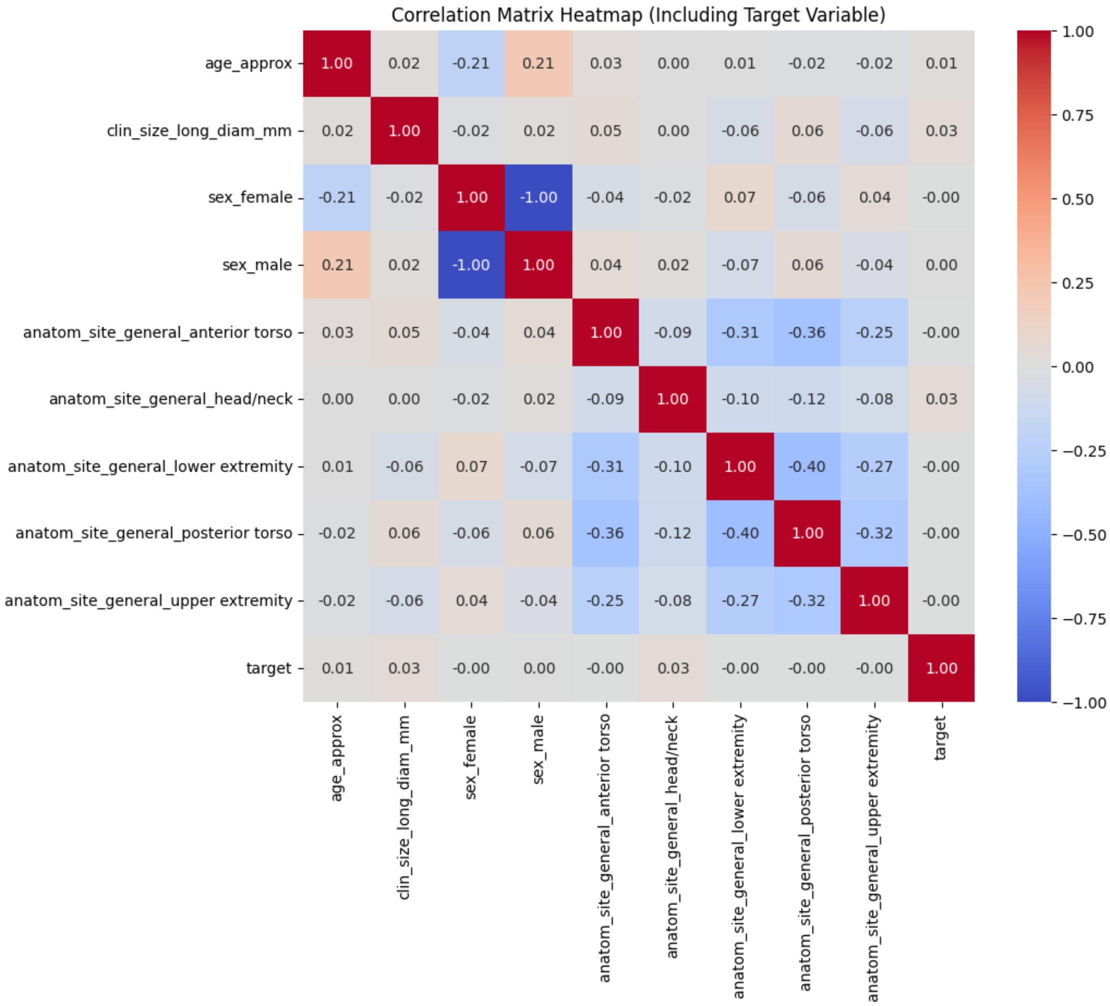
```
    # Fit method to learn the categories for encoding
    def fit(self, X, y=None):
        # Ensure input is a pandas DataFrame
        if not isinstance(X, pd.DataFrame):
            raise TypeError("Input must be a pandas DataFrame.")
        # Fit the encoder to categorical columns
        self.encoder.fit(X.select_dtypes(include=['object', 'category']))
        return self

    # Transform method to apply one-hot encoding
    def transform(self, X):
        # Ensure input is a pandas DataFrame
        if not isinstance(X, pd.DataFrame):
            raise TypeError("Input must be a pandas DataFrame.")

        # Transform categorical columns to one-hot encoding
        encoded_cols = self.encoder.transform(X.
↪select_dtypes(include=['object', 'category']))
        # Get the new column names after encoding
        new_columns = self.encoder.get_feature_names_out(X.
↪select_dtypes(include=['object', 'category']).columns)

        # Create a DataFrame for the encoded columns
        encode_df = pd.DataFrame(encoded_cols, columns=new_columns, index=X.
↪index)
        # Concatenate the original DataFrame (excluding categorical columns)␣
↪with the encoded DataFrame
        return pd.concat([X.select_dtypes(exclude=['object', 'category']),␣
↪encode_df], axis=1)

# Custom transformer for scaling numerical features
class NumericalScaler(BaseEstimator, TransformerMixin):
    def __init__(self):
        # Initialize the StandardScaler for scaling numerical features
        self.scaler = StandardScaler()

    # Fit method to learn the scaling parameters
    def fit(self, X, y=None):
        # Ensure input is a pandas DataFrame
        if not isinstance(X, pd.DataFrame):
            raise TypeError("Input must be a pandas DataFrame.")
        # Identify numerical columns
        num_cols = X.select_dtypes(include=['int64', 'float64']).columns
        # Fit the scaler to the numerical columns
        self.scaler.fit(X[num_cols])
        return self
```

```
    # Transform method to apply scaling
    def transform(self, X):
        # Ensure input is a pandas DataFrame
        if not isinstance(X, pd.DataFrame):
            raise TypeError("Input must be a pandas DataFrame.")

        # Identify numerical columns
        num_cols = X.select_dtypes(include=['int64', 'float64']).columns
        # Apply scaling to the numerical columns
        X[num_cols] = self.scaler.transform(X[num_cols])
        return X  # Return the scaled DataFrame

# Custom transformer for handling age approximation
class AgeApproxTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self  # No fitting required for this transformer

    # Transform method to round age approximations
    def transform(self, X):
        # Ensure input is a pandas DataFrame
        if not isinstance(X, pd.DataFrame):
            raise TypeError("Input must be a pandas DataFrame.")
        # Check if 'age_approx' is in the DataFrame
        if 'age_approx' in X.columns:
            # Round the age and convert to integer type
            X['age_approx'] = X['age_approx'].round().astype('Int64')
        return X  # Return the transformed DataFrame

# Create the complete pipeline for preprocessing
def create_pipeline() -> Pipeline:
    # Define a pipeline with the specified transformers
    pipeline = Pipeline(steps=[
        ('age_transformer', AgeApproxTransformer()),  # Age approximation
        ('missing_value_handler', MissingValueHandler()),  # Handling missing␣
 ↪values
        ('cat_encoder', OneHotEncoderTransformer()),  # One-hot encoding␣
 ↪categorical features
        ('num_scaler', NumericalScaler())  # Scaling all numerical features␣
 ↪(including encoded features)
    ])
    return pipeline  # Return the constructed pipeline
```

After week 4, I realized that applying StandardScaler to my dataset may not be the optimal choice for a neural network model. Instead, using MinMaxScaler is more appropriate, as it scales the data to a range of 0 to 1, which aligns better with the activation functions commonly used in neural networks. This adjustment ensures that the input features are normalized in a way that enhances the model's learning

efficiency and stability. Moving forward, this is one of the changes I will implement to improve the overall performance of my model.

[7]:
```python
# Load the training metadata from a CSV file



# Drop the 'target' and 'isic_id' columns to create the feature set
X = train_df.drop(columns=['target', 'isic_id'])

# Keep the 'target' and 'isic_id' columns in a separate DataFrame for later use
temp = train_df[['target', 'isic_id']]

# Create the preprocessing pipeline using the previously defined function
pipeline = create_pipeline()

try:
    # Fit the pipeline to the feature set and transform the data
    processed_X = pipeline.fit_transform(X)
except Exception as e:
    # Log any errors that occur during fitting and transformation
    print(f"Error occurred during pipeline processing: {e}")

# Concatenate the processed features with the target and ISIC ID columns
processed_df = pd.concat([processed_X, temp], axis=1)

# Calculate the correlation matrix, excluding the 'isic_id' column
correlation_matrix = processed_df.drop(columns=['isic_id']).corr()

# Set the size of the plot
plt.figure(figsize=(10, 8))

# Create the heatmap using seaborn
sns.heatmap(
    correlation_matrix,             # The correlation matrix to visualize
    annot=True,                     # Annotate each cell with the numeric value
    fmt=".2f",                      # Format the annotation to two decimal places
    cmap='coolwarm',                # Color map for the heatmap
    square=True                     # Ensure each cell is square-shaped
)

# Set the title for the plot
plt.title('Correlation Matrix Heatmap (Including Target Variable)')

# Display the plot
plt.show()
```



We can see that sex_female and sex_male are highly correlated, but I do not think it will significantly affect the accuracy of the neural network model because the relationship between these two features is binary and mutually exclusive. In this case, one being 1 automatically implies the other is 0. Neural networks are capable of learning such simple relationships efficiently without causing confusion or overfitting.

In the future, as a best practice, one of these features could be dropped without any loss of information, as retaining only sex_female (or sex_male) is sufficient to convey the same information. However, for interpretability and domain alignment, keeping both features might be beneficial depending on how the model's results are presented or used. This decision could also depend on how stakeholders prefer to view or analyze the results of the predictions.

4 4) Preprocess & Feature Engineer data

For metadata preprocessing, I will utilize a custom pipeline that includes handling missing values, one-hot encoding categorical variables, and scaling numerical features. This ensures that all metadata inputs are properly formatted for input into the neural network.

For image feature engineering, I will apply transformations such as resizing, rotation, and random cropping to the images. These transformations help improve model generalization by introducing variability in the training data, thereby reducing the risk of overfitting.

By combining these preprocessing steps, I aim to ensure that both the metadata and image inputs are in optimal condition for the neural network, leading to better model performance and robustness.

4.1 Handle data imbalance in training set

[8]:
```
# Assuming 'train' is your DataFrame with the target column 'target'
try:
    # Print class distribution before sampling
    print("Class Distribution Before Sampling (%):")
    display(train_df.target.value_counts(normalize=True) * 100)

    # Check if the 'target' column exists in the DataFrame
    if 'target' not in train_df.columns:
        raise KeyError("The 'target' column is not found in the DataFrame.")

    # Sampling process
    try:
        # Sample the majority class (0) with a fraction of 0.01
        majority_df = train_df.query("target == 0").sample(frac=0.01,␣
↪random_state=42)  # Fixed random seed for reproducibility

        # Sample the minority class (1) with a factor of 5.0, allowing␣
↪replacement
        minority_df = train_df.query("target == 1").sample(frac=5.0,␣
↪replace=True, random_state=42)

        # Combine the sampled data into a new balanced DataFrame
        train_balanced = pd.concat([majority_df, minority_df], axis=0).
↪sample(frac=1.0, random_state=42)  # Shuffle the combined DataFrame
    except ValueError as e:
        raise ValueError(f"Error during sampling: {e}")

    # Print class distribution after sampling
    print("\nClass Distribution After Sampling (%):")
```

```
    display(train_balanced.target.value_counts(normalize=True) * 100)

except Exception as e:
    print(f"An error occurred: {e}")
```

```
Class Distribution Before Sampling (%):

target
0    99.902045
1     0.097955
Name: proportion, dtype: float64


Class Distribution After Sampling (%):

target
0    67.105263
1    32.894737
Name: proportion, dtype: float64
```

As you can see, I have downsized the majority class and upsized the minority class to address the class imbalance in the dataset. This resampling strategy aims to create a more balanced distribution of classes, which can help the model better identify and classify minority class instances.

One important consideration is that this approach may still affect the model's ability to generalize to new image data and metadata. By artificially altering the class distribution, there is a risk of overfitting to the resampled data, especially if the model becomes too focused on the minority class. To mitigate this, I will employ strategies such as cross-validation, early stopping, and careful selection of evaluation metrics (e.g., AUROC, precision-recall) to ensure the model remains robust on unseen data.

4.2 Metadata Preprocessing Pipeline

The metadata prerpocessing pipeline includes:

-
-
-
-

[12]:
```
#seperate case id and target variable from dependable variables
pipeline = create_pipeline()
X_train = train_balanced.drop(columns=['isic_id','target'])
temp_train = train_balanced[['target','isic_id']]
```

```
train_processed_df = pd.concat([pipeline.
 ↪fit_transform(X_train),temp_train],axis=1)

# Process validation data
X_validation = validation_df.drop(columns=['isic_id', 'target'])
temp_validation = validation_df[['target', 'isic_id']]
validation_processed_df = pd.concat([pipeline.transform(X_validation),␣
 ↪temp_validation], axis=1)

# Process test data
X_test = test_df.drop(columns=['isic_id', 'target'])
temp_test = test_df[['target', 'isic_id']]
test_processed_df = pd.concat([pipeline.transform(X_test), temp_test], axis=1)

# Save the processed dataframes
train_processed_df.to_csv('../data/processed/processed-train-metadata.csv',␣
 ↪index=False)
validation_processed_df.to_csv('../data/processed/processed-validation-metadata.
 ↪csv', index=False)
test_processed_df.to_csv('../data/processed/processed-test-metadata.csv',␣
 ↪index=False)
```

4.3 Feature Engineer Image Data

I will create a custom dataset to store and preprocess the data, enabling efficient data loading and feature engineering for later use in the model. This approach ensures that the data is preprocessed consistently and allows for easy access during model training and evaluation.

4.3.1 Create Custom Dataset

[7]:
```
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
```

[8]:

```
# Feature Engineer for train,validation and test image data

def get_train_transform(resize_size=(224, 224), crop_size=128,␣
↪rotation_degree=10, normalize_means=(0.5, 0.5, 0.5), normalize_stds=(0.5, 0.
↪5, 0.5)):
```

```python
    """
    Returns the transformations for the training dataset, including data augmentation.

    Args:
        resize_size (tuple): The size to resize the image before cropping.
        crop_size (int): The size of the random crop.
        rotation_degree (int): Maximum degree for random rotation.
        normalize_means (tuple): Means for normalization.
        normalize_stds (tuple): Standard deviations for normalization.

    Returns:
        transforms.Compose: The composed transformations for the training set.
    """
    return transforms.Compose([
        transforms.Resize(resize_size),  # Resize to specified size
        transforms.RandomResizedCrop(crop_size, scale=(0.8, 1.0)),  # Random crop with scale
        transforms.RandomRotation(rotation_degree),  # Randomly rotate images
        transforms.ToTensor(),  # Convert image to PyTorch tensor
        transforms.Normalize(normalize_means, normalize_stds)  # Normalize with specified means and stds
    ])

def get_normal_transform(resize_size=(224, 224), normalize_means=(0.5, 0.5, 0.5), normalize_stds=(0.5, 0.5, 0.5)):
    """
    Returns the transformations for the validation/test dataset (without data augmentation).

    Args:
        resize_size (tuple): The size to resize the image.
        normalize_means (tuple): Means for normalization.
        normalize_stds (tuple): Standard deviations for normalization.

    Returns:
        transforms.Compose: The composed transformations for the validation/test set.
    """
    return transforms.Compose([
        transforms.Resize(resize_size),  # Resize to specified size
        transforms.ToTensor(),  # Convert image to PyTorch tensor
        transforms.Normalize(normalize_means, normalize_stds)  # Normalize with specified means and stds
    ])
```

5 Model Development

In this stage, I will use the resampled dataset to address size constraints and ensure efficient model development. The resampled dataset allows for faster computations while preserving the data's core characteristics, which is critical for iterative model development and evaluation.

I will develop three multi-input neural network models with slight variations in the image processing component. Each of these models will accept two inputs — image data and metadata — which will be processed independently before being combined for final prediction.

[11]:
```
device = "cuda" if torch.cuda.is_available() else "cpu" # this will deetct
```

5.1 Model Building

5.1.1 CNN

[5]:
```
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)
        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
```

5.1.2 Resnet

[6]:
```
class CustomImageFeatureResNet(nn.Module):
    def __init__(self, feature_input_size, pretrained=True):
        super(CustomImageFeatureResNet, self).__init__()

        # Load a pretrained ResNet model for image feature extraction (ResNet18␣
↪in this case)
        resnet = models.resnet18(pretrained=pretrained)  # Change to resnet50,␣
↪resnet101 as needed
        self.resnet = nn.Sequential(*list(resnet.children())[:-1])  # Remove␣
↪the final classification layer

        # The output of ResNet18's last conv layer is 512-dimensional (for␣
↪ResNet50, it would be 2048)
        self.fc_image = nn.Linear(resnet.fc.in_features, 512)  # Adjust if␣
↪using ResNet50

        # 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)  # For binary classification

    def forward(self, image, metadata):
        # Forward pass for the image through the ResNet (without the final␣
↪classification layer)
        x = self.resnet(image)  # ResNet feature extraction
        x = x.view(x.size(0), -1)  # Flatten the ResNet output
        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
```

5.1.3 EfficientNet

[3]:
```
class CustomImageFeatureEfficientNet(nn.Module):
    def __init__(self, feature_input_size, pretrained=True):
        super(CustomImageFeatureEfficientNet, self).__init__()

        # Load a pretrained EfficientNet model for image feature extraction␣
↪(EfficientNet-B0 in this case)
        efficientnet = models.efficientnet_b0(pretrained=pretrained)  # You can␣
↪change this to another EfficientNet version like B1 or B7
        self.efficientnet = nn.Sequential(*list(efficientnet.children())[:-1]) ␣
↪# Remove the final classification layer

        # The output of EfficientNet-B0's last conv layer is 1280-dimensional
        self.fc_image = nn.Linear(1280, 512)  # Reduce dimension to match your␣
↪custom architecture

        # 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)  # For binary classification

    def forward(self, image, metadata):
        # Forward pass for the image through EfficientNet (without the final␣
↪classification layer)
        x = self.efficientnet(image)  # EfficientNet feature extraction
        x = x.view(x.size(0), -1)  # Flatten the EfficientNet output
        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
```

5.1.4 Model Training

This cell contains the score function as well as the training and validation loop. The score function calculates the partial AUC-above-TPR, a key evaluation metric that focuses on the model's performance in high true positive rate regions. This is critical for ensuring that malignant lesions are correctly classified.

During the model training process, I implemented early stopping and model checkpointing to enhance performance and prevent overfitting. At each epoch, the model's validation loss is tracked, and if it achieves the lowest validation loss observed so far, the model is saved as the best model. This best-performing version will be used for later deployment, ensuring that only the most optimal and generalizable model is selected for real-world use. By doing so, I can ensure that the final deployed model achieves a balance between bias and variance while maintaining strong predictive performance on unseen data.

[13]:
```
# 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.
```

```python
        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
```

[8]:
```python
# 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,
    best_model_path: str,
    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}/{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_path)
            else:
                early_stopping_counter += 1

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

5.1.5 Ready DataLoader for training

[9]:
```
# Initialize the dataset
CNN_train_dataset = MultiInputDataset(hdf5_file='../data/raw/train_images.
 ↪hdf5', csv_file='../data/processed/processed-train-metadata1.csv',␣
 ↪transform=get_train_transform(resize_size=(128,128)))
CNN_val_dataset = MultiInputDataset(hdf5_file='../data/raw/validation_image.
 ↪hdf5', csv_file='../data/processed/processed-validation-metadata1.csv',␣
 ↪transform=get_normal_transform(resize_size=(128,128)))
# Create a DataLoader
CNN_train_dataloader = DataLoader(CNN_train_dataset,  batch_size=64,␣
 ↪shuffle=True)
CNN_val_dataloader = DataLoader(CNN_val_dataset,  batch_size=64, shuffle=True)
```

[10]:
```
# Initialize the dataset
resnet_train_dataset = MultiInputDataset(hdf5_file='../data/raw/train_images.
 ↪hdf5', csv_file='../data/processed/processed-train-metadata1.csv',␣
 ↪transform=get_train_transform(resize_size=(225,225)))
resnet_val_dataset = MultiInputDataset(hdf5_file='../data/raw/validation_image.
 ↪hdf5', csv_file='../data/processed/processed-validation-metadata1.csv',␣
 ↪transform=get_normal_transform(resize_size=(225,225)))
# Create a DataLoader
resnet_train_dataloader = DataLoader(resnet_train_dataset,  batch_size=64,␣
 ↪shuffle=True)
resnet_val_dataloader = DataLoader(resnet_val_dataset,  batch_size=64,␣
 ↪shuffle=True)
```

[11]:
```
# Initialize the dataset
effnet_train_dataset = MultiInputDataset(hdf5_file='../data/raw/train_images.
 ↪hdf5', csv_file='../data/processed/processed-train-metadata1.csv',␣
 ↪transform=get_train_transform(resize_size=(224,224)))
effnet_val_dataset = MultiInputDataset(hdf5_file='../data/raw/validation_image.
 ↪hdf5', csv_file='../data/processed/processed-validation-metadata1.csv',␣
 ↪transform=get_normal_transform(resize_size=(224,224)))
```

```
# Create a DataLoader
effnet_train_dataloader = DataLoader(effnet_train_dataset,  batch_size=64,␣
 ↪shuffle=True)
effnet_val_dataloader = DataLoader(effnet_val_dataset,  batch_size=64,␣
 ↪shuffle=True)
```

5.2 Hyperparameter Tuning

5.2.1 Model 1

[12]:
```
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
best_model_path = "best_model1.pth"
```

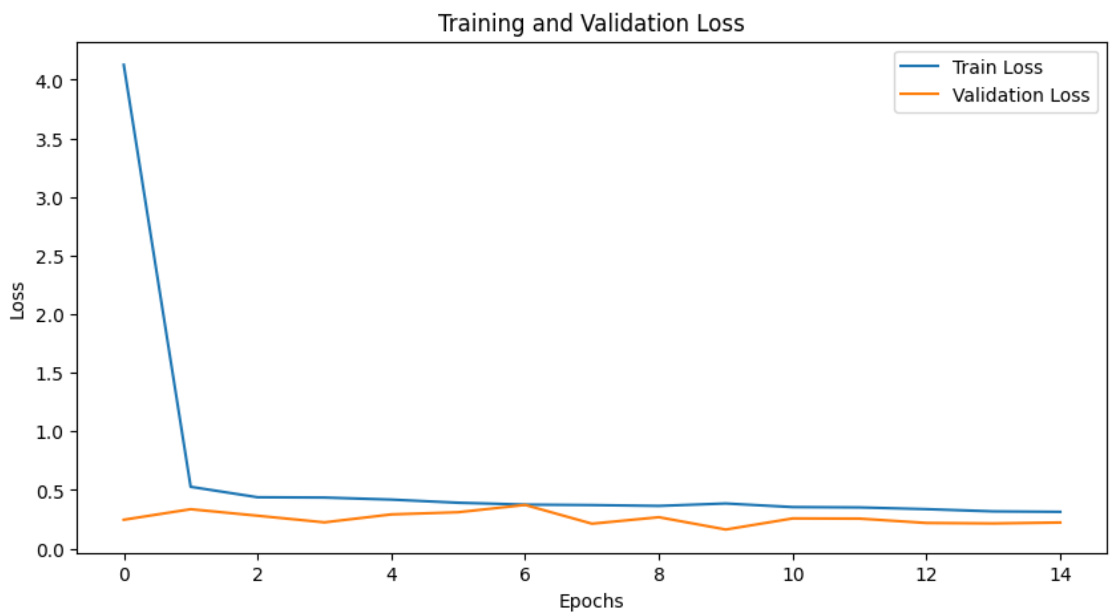
[13]:
```
train_and_validate(model1,CNN_train_dataloader, CNN_val_dataloader, criterion,␣
 ↪optimizer, epochs, device ,best_model_path)
```

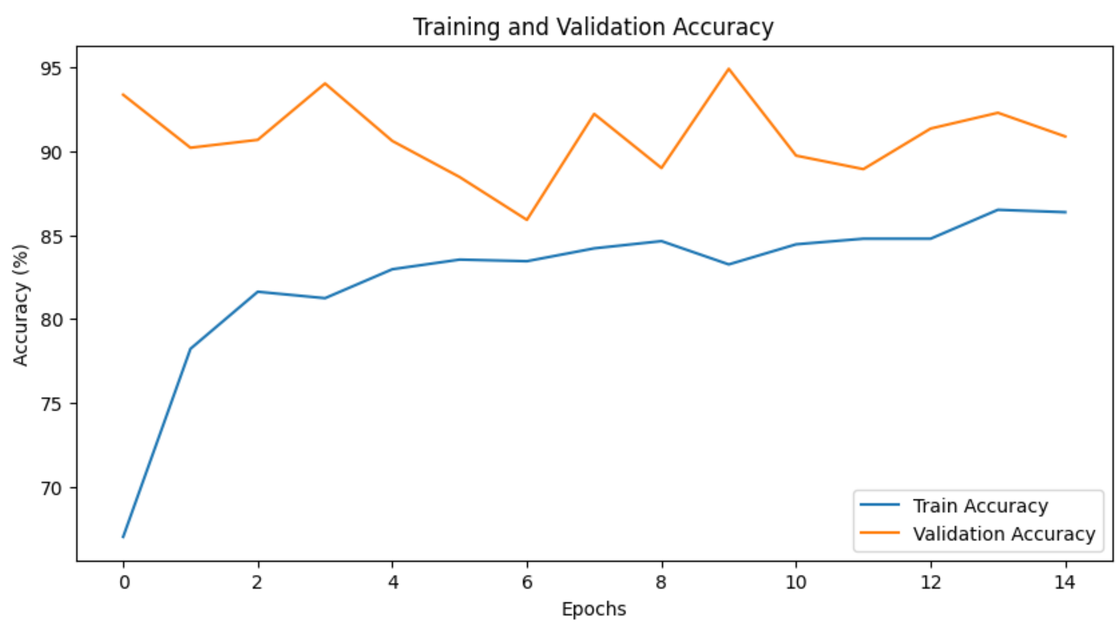
```
Epoch 1/20

Training Epoch 1: 100%|      | 33/33 [01:32<00:00,  2.80s/it,
train_loss=4.13]
Validating Epoch 1: 100%|      | 24/24 [00:28<00:00,  1.17s/it,
val_loss=0.246]

Epoch [0/20], Train Loss: 4.1273, Val Loss: 0.2455, Val Accuracy: 93.36%, Val
AUROC: 0.7198, Partial AUROC: 0.0416
Epoch 2/20

Training Epoch 2: 100%|      | 33/33 [01:26<00:00,  2.62s/it,
train_loss=0.527]
Validating Epoch 2: 100%|      | 24/24 [00:38<00:00,  1.62s/it,
val_loss=0.336]

Epoch [1/20], Train Loss: 0.5268, Val Loss: 0.3356, Val Accuracy: 90.20%, Val
AUROC: 0.8172, Partial AUROC: 0.0697
Epoch 3/20

Training Epoch 3: 100%|      | 33/33 [01:26<00:00,  2.64s/it,
train_loss=0.438]
Validating Epoch 3: 100%|      | 24/24 [00:28<00:00,  1.18s/it,
val_loss=0.28]

Epoch [2/20], Train Loss: 0.4376, Val Loss: 0.2800, Val Accuracy: 90.67%, Val
```

```
AUROC: 0.8300, Partial AUROC: 0.0776
Epoch 4/20

Training Epoch 4: 100%|     | 33/33 [01:28<00:00,  2.69s/it,
train_loss=0.435]
Validating Epoch 4: 100%|     | 24/24 [00:27<00:00,  1.17s/it,
val_loss=0.223]

Epoch [3/20], Train Loss: 0.4350, Val Loss: 0.2235, Val Accuracy: 94.03%, Val
AUROC: 0.8185, Partial AUROC: 0.0729
Epoch 5/20

Training Epoch 5: 100%|     | 33/33 [01:36<00:00,  2.92s/it,
train_loss=0.418]
Validating Epoch 5: 100%|     | 24/24 [00:27<00:00,  1.17s/it,
val_loss=0.291]

Epoch [4/20], Train Loss: 0.4175, Val Loss: 0.2912, Val Accuracy: 90.60%, Val
AUROC: 0.8401, Partial AUROC: 0.0843
Epoch 6/20

Training Epoch 6: 100%|     | 33/33 [01:31<00:00,  2.79s/it,
train_loss=0.391]
Validating Epoch 6: 100%|     | 24/24 [00:27<00:00,  1.16s/it,
val_loss=0.31]

Epoch [5/20], Train Loss: 0.3907, Val Loss: 0.3098, Val Accuracy: 88.46%, Val
AUROC: 0.8516, Partial AUROC: 0.0934
Epoch 7/20

Training Epoch 7: 100%|     | 33/33 [01:37<00:00,  2.96s/it,
train_loss=0.375]
Validating Epoch 7: 100%|     | 24/24 [00:27<00:00,  1.16s/it,
val_loss=0.372]

Epoch [6/20], Train Loss: 0.3750, Val Loss: 0.3721, Val Accuracy: 85.91%, Val
AUROC: 0.8669, Partial AUROC: 0.1072
Epoch 8/20

Training Epoch 8: 100%|     | 33/33 [01:26<00:00,  2.62s/it,
train_loss=0.371]
Validating Epoch 8: 100%|     | 24/24 [00:28<00:00,  1.17s/it,
val_loss=0.212]

Epoch [7/20], Train Loss: 0.3710, Val Loss: 0.2119, Val Accuracy: 92.21%, Val
AUROC: 0.8453, Partial AUROC: 0.0894
Epoch 9/20

Training Epoch 9: 100%|     | 33/33 [01:26<00:00,  2.61s/it,
train_loss=0.364]
Validating Epoch 9: 100%|     | 24/24 [00:28<00:00,  1.17s/it,
val_loss=0.267]
```

```
Epoch [8/20], Train Loss: 0.3638, Val Loss: 0.2673, Val Accuracy: 88.99%, Val
AUROC: 0.8541, Partial AUROC: 0.0993
Epoch 10/20

Training Epoch 10: 100%|     | 33/33 [01:38<00:00,  2.97s/it,
train_loss=0.384]
Validating Epoch 10: 100%|     | 24/24 [00:27<00:00,  1.17s/it,
val_loss=0.162]

Epoch [9/20], Train Loss: 0.3844, Val Loss: 0.1621, Val Accuracy: 94.90%, Val
AUROC: 0.8588, Partial AUROC: 0.0990
Epoch 11/20

Training Epoch 11: 100%|     | 33/33 [01:35<00:00,  2.90s/it,
train_loss=0.354]
Validating Epoch 11: 100%|     | 24/24 [00:28<00:00,  1.17s/it,
val_loss=0.257]

Epoch [10/20], Train Loss: 0.3538, Val Loss: 0.2568, Val Accuracy: 89.73%, Val
AUROC: 0.8791, Partial AUROC: 0.1152
Epoch 12/20

Training Epoch 12: 100%|     | 33/33 [01:27<00:00,  2.65s/it,
train_loss=0.351]
Validating Epoch 12: 100%|     | 24/24 [00:39<00:00,  1.65s/it,
val_loss=0.255]

Epoch [11/20], Train Loss: 0.3506, Val Loss: 0.2552, Val Accuracy: 88.93%, Val
AUROC: 0.8460, Partial AUROC: 0.0960
Epoch 13/20

Training Epoch 13: 100%|     | 33/33 [01:27<00:00,  2.67s/it,
train_loss=0.336]
Validating Epoch 13: 100%|     | 24/24 [00:28<00:00,  1.20s/it,
val_loss=0.218]

Epoch [12/20], Train Loss: 0.3362, Val Loss: 0.2182, Val Accuracy: 91.34%, Val
AUROC: 0.8635, Partial AUROC: 0.1033
Epoch 14/20

Training Epoch 14: 100%|     | 33/33 [01:28<00:00,  2.70s/it,
train_loss=0.317]
Validating Epoch 14: 100%|     | 24/24 [00:29<00:00,  1.21s/it,
val_loss=0.214]

Epoch [13/20], Train Loss: 0.3170, Val Loss: 0.2141, Val Accuracy: 92.28%, Val
AUROC: 0.8681, Partial AUROC: 0.1019
Epoch 15/20

Training Epoch 15: 100%|     | 33/33 [01:38<00:00,  2.98s/it,
train_loss=0.313]
Validating Epoch 15: 100%|     | 24/24 [00:28<00:00,  1.20s/it,
val_loss=0.222]
```

```
Epoch [14/20], Train Loss: 0.3130, Val Loss: 0.2220, Val Accuracy: 90.87%, Val
AUROC: 0.8661, Partial AUROC: 0.1032
Early stopping triggered at epoch 14
Best Epoch: 10, Best Validation Loss: 0.1621
Training Complete
```






```
Classification Report:
```

```
              precision    recall  f1-score   support

     Class 0       0.98      0.92      0.95      1431
     Class 1       0.25      0.64      0.36        59

    accuracy                           0.91      1490
   macro avg       0.62      0.78      0.65      1490
weighted avg       0.96      0.91      0.93      1490
```

[13]:
```
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)
)
```

5.3 Model 2

[14]:
```python
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
best_model_path = "best_model2.pth"
```

[15]:
```python
train_and_validate(model2,CNN_train_dataloader, CNN_val_dataloader, criterion,␣
 ↪optimizer, epochs, device,best_model_path )
```

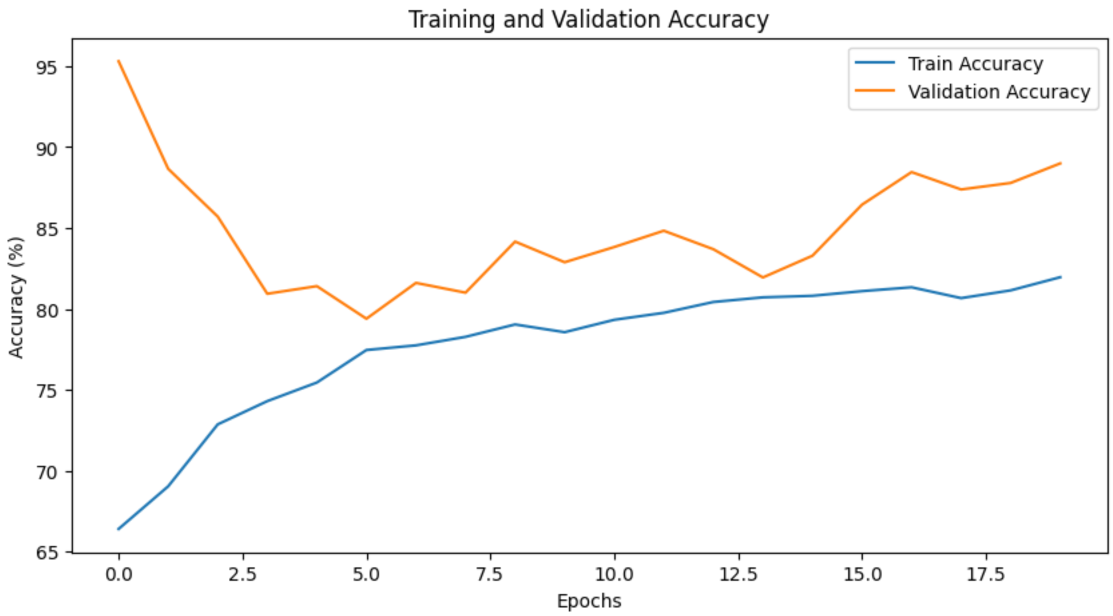
```
Epoch 1/20

Training Epoch 1: 100%|      | 33/33 [01:28<00:00,  2.69s/it,
train_loss=0.62]
```

```
Validating Epoch 1: 100%|     | 24/24 [00:28<00:00,  1.18s/it,
val_loss=0.578]

Epoch [0/20], Train Loss: 0.6199, Val Loss: 0.5776, Val Accuracy: 95.30%, Val
AUROC: 0.6968, Partial AUROC: 0.0424
Epoch 2/20

Training Epoch 2: 100%|     | 33/33 [01:22<00:00,  2.49s/it,
train_loss=0.59]
Validating Epoch 2: 100%|     | 24/24 [00:37<00:00,  1.58s/it,
val_loss=0.534]

Epoch [1/20], Train Loss: 0.5904, Val Loss: 0.5338, Val Accuracy: 88.66%, Val
AUROC: 0.7621, Partial AUROC: 0.0597
Epoch 3/20

Training Epoch 3: 100%|     | 33/33 [01:22<00:00,  2.50s/it,
train_loss=0.567]
Validating Epoch 3: 100%|     | 24/24 [00:28<00:00,  1.17s/it,
val_loss=0.507]

Epoch [2/20], Train Loss: 0.5667, Val Loss: 0.5072, Val Accuracy: 85.70%, Val
AUROC: 0.7806, Partial AUROC: 0.0627
Epoch 4/20

Training Epoch 4: 100%|     | 33/33 [01:23<00:00,  2.52s/it,
train_loss=0.552]
Validating Epoch 4: 100%|     | 24/24 [00:28<00:00,  1.18s/it,
val_loss=0.521]

Epoch [3/20], Train Loss: 0.5519, Val Loss: 0.5210, Val Accuracy: 80.94%, Val
AUROC: 0.7852, Partial AUROC: 0.0614
Epoch 5/20

Training Epoch 5: 100%|     | 33/33 [01:35<00:00,  2.88s/it,
train_loss=0.53]
Validating Epoch 5: 100%|     | 24/24 [00:28<00:00,  1.17s/it,
val_loss=0.507]

Epoch [4/20], Train Loss: 0.5305, Val Loss: 0.5073, Val Accuracy: 81.41%, Val
AUROC: 0.7933, Partial AUROC: 0.0634
Epoch 6/20

Training Epoch 6: 100%|     | 33/33 [01:28<00:00,  2.68s/it,
train_loss=0.508]
Validating Epoch 6: 100%|     | 24/24 [00:28<00:00,  1.21s/it,
val_loss=0.513]

Epoch [5/20], Train Loss: 0.5075, Val Loss: 0.5125, Val Accuracy: 79.40%, Val
AUROC: 0.7999, Partial AUROC: 0.0662
Epoch 7/20

Training Epoch 7: 100%|     | 33/33 [01:24<00:00,  2.55s/it,
train_loss=0.5]
```

```
Validating Epoch 7: 100%|     | 24/24 [00:29<00:00,  1.21s/it,
val_loss=0.485]

Epoch [6/20], Train Loss: 0.5004, Val Loss: 0.4854, Val Accuracy: 81.61%, Val
AUROC: 0.8034, Partial AUROC: 0.0688
Epoch 8/20

Training Epoch 8: 100%|     | 33/33 [01:35<00:00,  2.91s/it,
train_loss=0.484]
Validating Epoch 8: 100%|     | 24/24 [00:28<00:00,  1.18s/it,
val_loss=0.478]

Epoch [7/20], Train Loss: 0.4840, Val Loss: 0.4780, Val Accuracy: 81.01%, Val
AUROC: 0.8071, Partial AUROC: 0.0681
Epoch 9/20

Training Epoch 9: 100%|     | 33/33 [01:23<00:00,  2.53s/it,
train_loss=0.476]
Validating Epoch 9: 100%|     | 24/24 [00:28<00:00,  1.18s/it,
val_loss=0.444]

Epoch [8/20], Train Loss: 0.4759, Val Loss: 0.4438, Val Accuracy: 84.16%, Val
AUROC: 0.8128, Partial AUROC: 0.0727
Epoch 10/20

Training Epoch 10: 100%|     | 33/33 [01:23<00:00,  2.53s/it,
train_loss=0.474]
Validating Epoch 10: 100%|     | 24/24 [00:28<00:00,  1.18s/it,
val_loss=0.455]

Epoch [9/20], Train Loss: 0.4739, Val Loss: 0.4552, Val Accuracy: 82.89%, Val
AUROC: 0.8160, Partial AUROC: 0.0731
Epoch 11/20

Training Epoch 11: 100%|     | 33/33 [01:39<00:00,  3.01s/it,
train_loss=0.469]
Validating Epoch 11: 100%|     | 24/24 [00:28<00:00,  1.17s/it,
val_loss=0.448]

Epoch [10/20], Train Loss: 0.4689, Val Loss: 0.4483, Val Accuracy: 83.83%, Val
AUROC: 0.8204, Partial AUROC: 0.0767
Epoch 12/20

Training Epoch 12: 100%|     | 33/33 [01:22<00:00,  2.49s/it,
train_loss=0.458]
Validating Epoch 12: 100%|     | 24/24 [00:28<00:00,  1.17s/it,
val_loss=0.425]

Epoch [11/20], Train Loss: 0.4580, Val Loss: 0.4253, Val Accuracy: 84.83%, Val
AUROC: 0.8240, Partial AUROC: 0.0777
Epoch 13/20

Training Epoch 13: 100%|     | 33/33 [01:33<00:00,  2.84s/it,
train_loss=0.453]
```

```
Validating Epoch 13: 100%|     | 24/24 [00:28<00:00,  1.18s/it,
val_loss=0.438]

Epoch [12/20], Train Loss: 0.4533, Val Loss: 0.4380, Val Accuracy: 83.69%, Val
AUROC: 0.8244, Partial AUROC: 0.0775
Epoch 14/20

Training Epoch 14: 100%|     | 33/33 [01:22<00:00,  2.51s/it,
train_loss=0.447]
Validating Epoch 14: 100%|     | 24/24 [00:27<00:00,  1.17s/it,
val_loss=0.46]

Epoch [13/20], Train Loss: 0.4468, Val Loss: 0.4599, Val Accuracy: 81.95%, Val
AUROC: 0.8281, Partial AUROC: 0.0793
Epoch 15/20

Training Epoch 15: 100%|     | 33/33 [01:22<00:00,  2.49s/it,
train_loss=0.439]
Validating Epoch 15: 100%|     | 24/24 [00:28<00:00,  1.17s/it,
val_loss=0.433]

Epoch [14/20], Train Loss: 0.4393, Val Loss: 0.4329, Val Accuracy: 83.29%, Val
AUROC: 0.8274, Partial AUROC: 0.0782
Epoch 16/20

Training Epoch 16: 100%|     | 33/33 [01:34<00:00,  2.86s/it,
train_loss=0.44]
Validating Epoch 16: 100%|     | 24/24 [00:31<00:00,  1.33s/it,
val_loss=0.398]

Epoch [15/20], Train Loss: 0.4400, Val Loss: 0.3978, Val Accuracy: 86.44%, Val
AUROC: 0.8318, Partial AUROC: 0.0862
Epoch 17/20

Training Epoch 17: 100%|     | 33/33 [01:23<00:00,  2.52s/it,
train_loss=0.43]
Validating Epoch 17: 100%|     | 24/24 [00:27<00:00,  1.16s/it,
val_loss=0.363]

Epoch [16/20], Train Loss: 0.4300, Val Loss: 0.3630, Val Accuracy: 88.46%, Val
AUROC: 0.8365, Partial AUROC: 0.0855
Epoch 18/20

Training Epoch 18: 100%|     | 33/33 [01:32<00:00,  2.82s/it,
train_loss=0.437]
Validating Epoch 18: 100%|     | 24/24 [00:27<00:00,  1.16s/it,
val_loss=0.372]

Epoch [17/20], Train Loss: 0.4373, Val Loss: 0.3721, Val Accuracy: 87.38%, Val
AUROC: 0.8355, Partial AUROC: 0.0844
Epoch 19/20

Training Epoch 19: 100%|     | 33/33 [01:22<00:00,  2.50s/it,
train_loss=0.433]
```

```
Validating Epoch 19: 100%|      | 24/24 [00:27<00:00,  1.16s/it,
val_loss=0.383]

Epoch [18/20], Train Loss: 0.4329, Val Loss: 0.3827, Val Accuracy: 87.79%, Val
AUROC: 0.8390, Partial AUROC: 0.0877
Epoch 20/20

Training Epoch 20: 100%|      | 33/33 [01:22<00:00,  2.51s/it,
train_loss=0.418]
Validating Epoch 20: 100%|      | 24/24 [00:28<00:00,  1.17s/it,
val_loss=0.343]

Epoch [19/20], Train Loss: 0.4184, Val Loss: 0.3431, Val Accuracy: 88.99%, Val
AUROC: 0.8399, Partial AUROC: 0.0882
Best Epoch: 20, Best Validation Loss: 0.3431
Training Complete
```




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

     Class 0       0.98      0.90      0.94      1431
     Class 1       0.21      0.63      0.31        59

    accuracy                           0.89      1490
   macro avg       0.59      0.76      0.63      1490
weighted avg       0.95      0.89      0.92      1490
```

[15]:
```
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)
```

```
)
```

5.4 Model 3

[16]:
```
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
best_model_path = "best_model3.pth"
```

[17]:
```
CNN_train_dataloader = DataLoader(CNN_train_dataset, batch_size=batch_size,␣
 ↪shuffle=True)
CNN_val_dataloader = DataLoader(CNN_val_dataset, batch_size=batch_size,␣
 ↪shuffle=True)
```

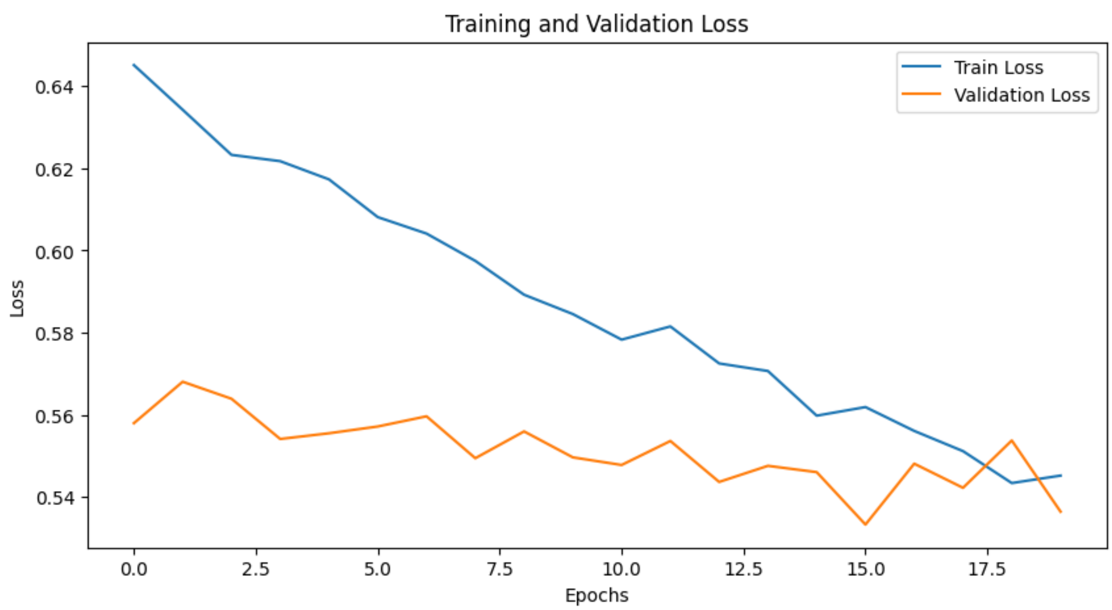
[18]:
```
train_and_validate(model3,CNN_train_dataloader, CNN_val_dataloader, criterion,␣
 ↪optimizer, epochs, device, best_model_path )
```

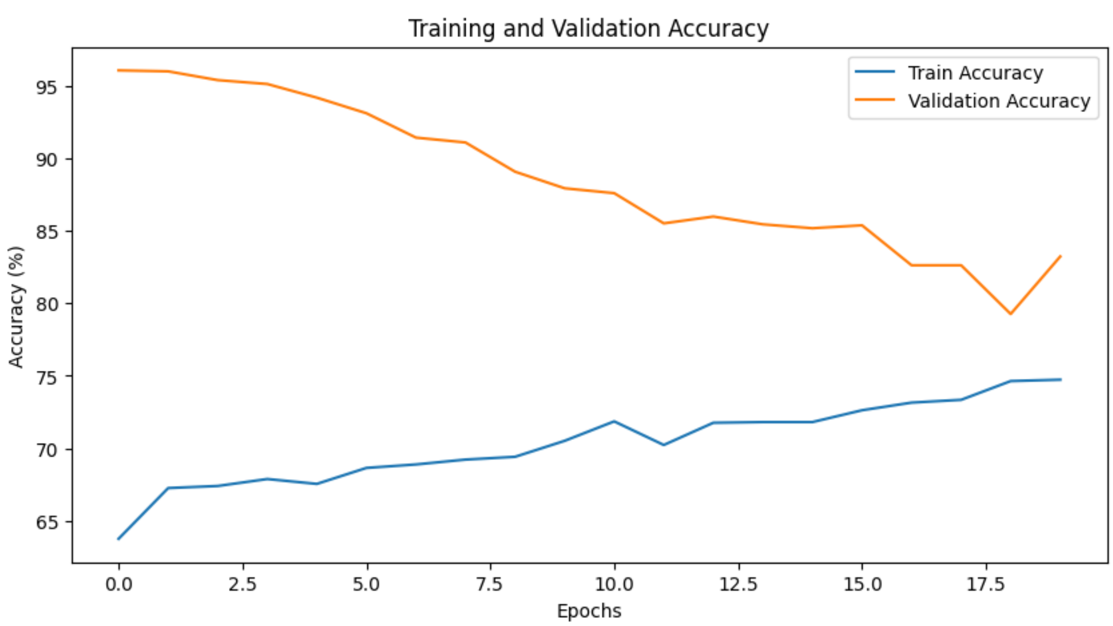
```
Epoch 1/20

Training Epoch 1: 100%|     | 66/66 [01:23<00:00,  1.27s/it,
train_loss=0.645]
Validating Epoch 1: 100%|     | 47/47 [00:23<00:00,  1.96it/s,
val_loss=0.558]

Epoch [0/20], Train Loss: 0.6451, Val Loss: 0.5580, Val Accuracy: 96.04%, Val
AUROC: 0.5562, Partial AUROC: 0.0358
Epoch 2/20

Training Epoch 2: 100%|     | 66/66 [01:17<00:00,  1.17s/it,
train_loss=0.634]
Validating Epoch 2: 100%|     | 47/47 [00:23<00:00,  2.02it/s,
val_loss=0.568]

Epoch [1/20], Train Loss: 0.6342, Val Loss: 0.5681, Val Accuracy: 95.97%, Val
AUROC: 0.6225, Partial AUROC: 0.0459
Epoch 3/20

Training Epoch 3: 100%|     | 66/66 [01:10<00:00,  1.06s/it,
train_loss=0.623]
Validating Epoch 3: 100%|     | 47/47 [00:21<00:00,  2.20it/s,
val_loss=0.564]
```

```
Epoch [2/20], Train Loss: 0.6233, Val Loss: 0.5639, Val Accuracy: 95.37%, Val
AUROC: 0.6612, Partial AUROC: 0.0525
Epoch 4/20

Training Epoch 4: 100%|     | 66/66 [01:22<00:00,  1.25s/it,
train_loss=0.622]
Validating Epoch 4: 100%|     | 47/47 [00:23<00:00,  2.03it/s,
val_loss=0.554]

Epoch [3/20], Train Loss: 0.6217, Val Loss: 0.5541, Val Accuracy: 95.10%, Val
AUROC: 0.6861, Partial AUROC: 0.0552
Epoch 5/20

Training Epoch 5: 100%|     | 66/66 [01:07<00:00,  1.02s/it,
train_loss=0.617]
Validating Epoch 5: 100%|     | 47/47 [00:23<00:00,  2.04it/s,
val_loss=0.556]

Epoch [4/20], Train Loss: 0.6173, Val Loss: 0.5555, Val Accuracy: 94.16%, Val
AUROC: 0.7074, Partial AUROC: 0.0583
Epoch 6/20

Training Epoch 6: 100%|     | 66/66 [01:09<00:00,  1.06s/it,
train_loss=0.608]
Validating Epoch 6: 100%|     | 47/47 [00:21<00:00,  2.15it/s,
val_loss=0.557]

Epoch [5/20], Train Loss: 0.6081, Val Loss: 0.5572, Val Accuracy: 93.09%, Val
AUROC: 0.7227, Partial AUROC: 0.0602
Epoch 7/20

Training Epoch 7: 100%|     | 66/66 [01:27<00:00,  1.32s/it,
train_loss=0.604]
Validating Epoch 7: 100%|     | 47/47 [00:22<00:00,  2.05it/s,
val_loss=0.56]

Epoch [6/20], Train Loss: 0.6041, Val Loss: 0.5597, Val Accuracy: 91.41%, Val
AUROC: 0.7311, Partial AUROC: 0.0597
Epoch 8/20

Training Epoch 8: 100%|     | 66/66 [01:18<00:00,  1.19s/it,
train_loss=0.597]
Validating Epoch 8: 100%|     | 47/47 [00:21<00:00,  2.21it/s,
val_loss=0.549]

Epoch [7/20], Train Loss: 0.5975, Val Loss: 0.5495, Val Accuracy: 91.07%, Val
AUROC: 0.7372, Partial AUROC: 0.0597
Epoch 9/20

Training Epoch 9: 100%|     | 66/66 [01:12<00:00,  1.10s/it,
train_loss=0.589]
Validating Epoch 9: 100%|     | 47/47 [00:23<00:00,  2.02it/s,
val_loss=0.556]
```

```
Epoch [8/20], Train Loss: 0.5892, Val Loss: 0.5560, Val Accuracy: 89.06%, Val
AUROC: 0.7492, Partial AUROC: 0.0607
Epoch 10/20

Training Epoch 10: 100%|      | 66/66 [01:10<00:00,  1.07s/it,
train_loss=0.585]
Validating Epoch 10: 100%|      | 47/47 [00:23<00:00,  2.00it/s,
val_loss=0.55]

Epoch [9/20], Train Loss: 0.5845, Val Loss: 0.5497, Val Accuracy: 87.92%, Val
AUROC: 0.7512, Partial AUROC: 0.0594
Epoch 11/20

Training Epoch 11: 100%|      | 66/66 [01:18<00:00,  1.19s/it,
train_loss=0.578]
Validating Epoch 11: 100%|      | 47/47 [00:23<00:00,  2.02it/s,
val_loss=0.548]

Epoch [10/20], Train Loss: 0.5783, Val Loss: 0.5478, Val Accuracy: 87.58%, Val
AUROC: 0.7573, Partial AUROC: 0.0606
Epoch 12/20

Training Epoch 12: 100%|      | 66/66 [01:07<00:00,  1.03s/it,
train_loss=0.582]
Validating Epoch 12: 100%|      | 47/47 [00:23<00:00,  2.01it/s,
val_loss=0.554]

Epoch [11/20], Train Loss: 0.5815, Val Loss: 0.5536, Val Accuracy: 85.50%, Val
AUROC: 0.7633, Partial AUROC: 0.0600
Epoch 13/20

Training Epoch 13: 100%|      | 66/66 [01:09<00:00,  1.05s/it,
train_loss=0.573]
Validating Epoch 13: 100%|      | 47/47 [00:24<00:00,  1.95it/s,
val_loss=0.544]

Epoch [12/20], Train Loss: 0.5725, Val Loss: 0.5437, Val Accuracy: 85.97%, Val
AUROC: 0.7637, Partial AUROC: 0.0598
Epoch 14/20

Training Epoch 14: 100%|      | 66/66 [01:27<00:00,  1.32s/it,
train_loss=0.571]
Validating Epoch 14: 100%|      | 47/47 [00:23<00:00,  2.02it/s,
val_loss=0.548]

Epoch [13/20], Train Loss: 0.5707, Val Loss: 0.5476, Val Accuracy: 85.44%, Val
AUROC: 0.7719, Partial AUROC: 0.0614
Epoch 15/20

Training Epoch 15: 100%|      | 66/66 [01:08<00:00,  1.04s/it,
train_loss=0.56]
Validating Epoch 15: 100%|      | 47/47 [00:23<00:00,  2.03it/s,
val_loss=0.546]
```

```
Epoch [14/20], Train Loss: 0.5598, Val Loss: 0.5461, Val Accuracy: 85.17%, Val
AUROC: 0.7738, Partial AUROC: 0.0604
Epoch 16/20

Training Epoch 16: 100%|     | 66/66 [01:08<00:00,  1.04s/it,
train_loss=0.562]
Validating Epoch 16: 100%|     | 47/47 [00:23<00:00,  2.04it/s,
val_loss=0.533]

Epoch [15/20], Train Loss: 0.5619, Val Loss: 0.5333, Val Accuracy: 85.37%, Val
AUROC: 0.7732, Partial AUROC: 0.0599
Epoch 17/20

Training Epoch 17: 100%|     | 66/66 [01:23<00:00,  1.26s/it,
train_loss=0.556]
Validating Epoch 17: 100%|     | 47/47 [00:23<00:00,  2.03it/s,
val_loss=0.548]

Epoch [16/20], Train Loss: 0.5561, Val Loss: 0.5481, Val Accuracy: 82.62%, Val
AUROC: 0.7807, Partial AUROC: 0.0623
Epoch 18/20

Training Epoch 18: 100%|     | 66/66 [01:10<00:00,  1.07s/it,
train_loss=0.551]
Validating Epoch 18: 100%|     | 47/47 [00:23<00:00,  2.03it/s,
val_loss=0.542]

Epoch [17/20], Train Loss: 0.5512, Val Loss: 0.5423, Val Accuracy: 82.62%, Val
AUROC: 0.7799, Partial AUROC: 0.0609
Epoch 19/20

Training Epoch 19: 100%|     | 66/66 [01:08<00:00,  1.03s/it,
train_loss=0.543]
Validating Epoch 19: 100%|     | 47/47 [00:22<00:00,  2.05it/s,
val_loss=0.554]

Epoch [18/20], Train Loss: 0.5434, Val Loss: 0.5538, Val Accuracy: 79.26%, Val
AUROC: 0.7839, Partial AUROC: 0.0619
Epoch 20/20

Training Epoch 20: 100%|     | 66/66 [01:14<00:00,  1.13s/it,
train_loss=0.545]
Validating Epoch 20: 100%|     | 47/47 [00:23<00:00,  1.99it/s,
val_loss=0.536]

Epoch [19/20], Train Loss: 0.5452, Val Loss: 0.5365, Val Accuracy: 83.22%, Val
AUROC: 0.7869, Partial AUROC: 0.0620
Best Epoch: 16, Best Validation Loss: 0.5333
Training Complete
```






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

     Class 0       0.98      0.84      0.91      1431
     Class 1       0.14      0.64      0.23        59

    accuracy                           0.83      1490
```

```
   macro avg       0.56      0.74      0.57      1490
weighted avg       0.95      0.83      0.88      1490
```

[18]:
```
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)
)
```

5.5 Model 4

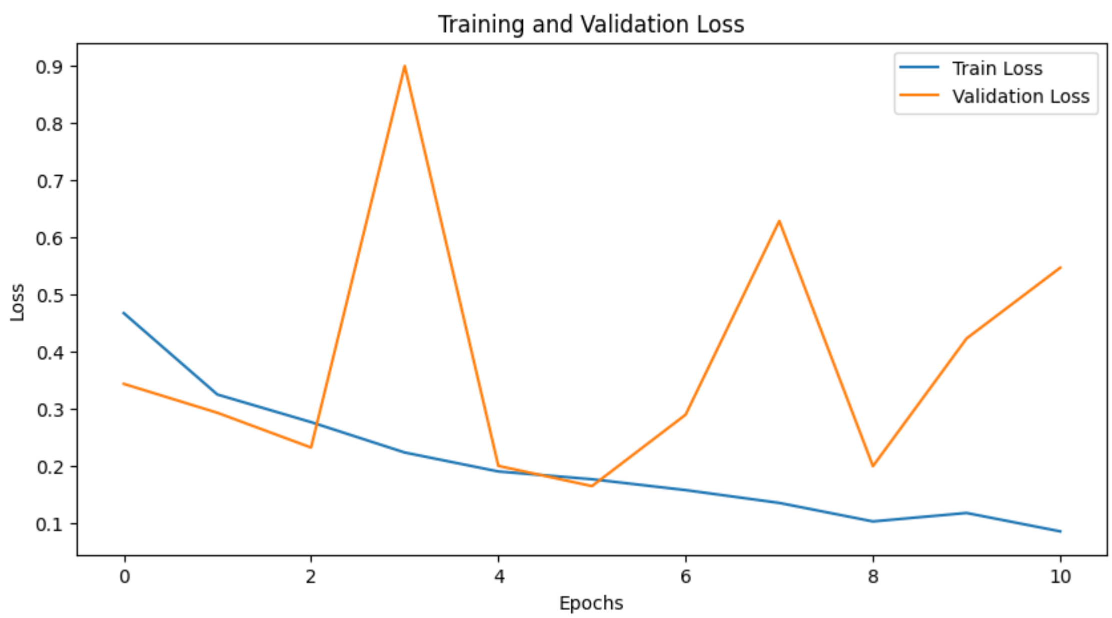
[19]:
```python
model4 = CustomImageFeatureResNet(feature_input_size=9)  # Assuming 9 features␣
 ↪for metadata
model4.to(device)
# Initialize optimizer
optimizer = optim.Adam(model4.parameters(), lr=0.001)
# Define the loss function with the class weights
criterion = nn.BCELoss()  # Binary classification loss
# Set the number of epochs
epochs = 20
best_model_path = "best_model4.pth"
```

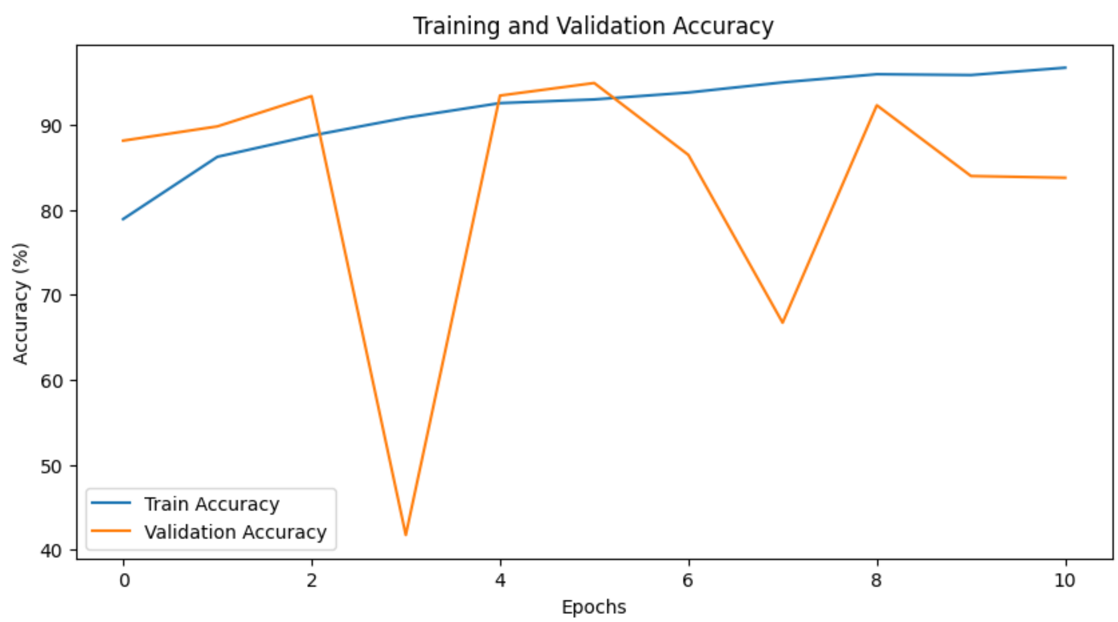
```
/home/jupyter-sohka/.local/lib/python3.10/site-
packages/torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
/home/jupyter-sohka/.local/lib/python3.10/site-
packages/torchvision/models/_utils.py:223: 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)
```

[20]:
```
train_and_validate(model4,resnet_train_dataloader, resnet_val_dataloader,␣
↪criterion, optimizer, epochs, device, best_model_path )
```

```
Epoch 1/20

Training Epoch 1: 100%|     | 33/33 [01:58<00:00,  3.58s/it,
train_loss=0.467]
Validating Epoch 1: 100%|     | 24/24 [01:23<00:00,  3.49s/it,
val_loss=0.344]

Epoch [0/20], Train Loss: 0.4672, Val Loss: 0.3437, Val Accuracy: 88.12%, Val
AUROC: 0.8037, Partial AUROC: 0.0793
Epoch 2/20

Training Epoch 2: 100%|     | 33/33 [01:44<00:00,  3.17s/it,
train_loss=0.325]
Validating Epoch 2: 100%|     | 24/24 [01:28<00:00,  3.69s/it,
val_loss=0.293]

Epoch [1/20], Train Loss: 0.3251, Val Loss: 0.2932, Val Accuracy: 89.80%, Val
AUROC: 0.7797, Partial AUROC: 0.0647
Epoch 3/20

Training Epoch 3: 100%|     | 33/33 [01:55<00:00,  3.51s/it,
train_loss=0.277]
Validating Epoch 3: 100%|     | 24/24 [01:33<00:00,  3.90s/it,
val_loss=0.233]

Epoch [2/20], Train Loss: 0.2769, Val Loss: 0.2325, Val Accuracy: 93.36%, Val
AUROC: 0.8221, Partial AUROC: 0.0836
Epoch 4/20

Training Epoch 4: 100%|     | 33/33 [01:45<00:00,  3.20s/it,
train_loss=0.224]
Validating Epoch 4: 100%|     | 24/24 [01:34<00:00,  3.93s/it,
val_loss=0.899]

Epoch [3/20], Train Loss: 0.2239, Val Loss: 0.8988, Val Accuracy: 41.74%, Val
AUROC: 0.8062, Partial AUROC: 0.0817
Epoch 5/20

Training Epoch 5: 100%|     | 33/33 [01:44<00:00,  3.17s/it,
train_loss=0.191]
Validating Epoch 5: 100%|     | 24/24 [01:39<00:00,  4.13s/it,
val_loss=0.201]

Epoch [4/20], Train Loss: 0.1907, Val Loss: 0.2005, Val Accuracy: 93.42%, Val
AUROC: 0.8241, Partial AUROC: 0.0807
Epoch 6/20

Training Epoch 6: 100%|     | 33/33 [01:45<00:00,  3.20s/it,
train_loss=0.177]
```

```
Validating Epoch 6: 100%|     | 24/24 [01:34<00:00,  3.95s/it,
val_loss=0.165]

Epoch [5/20], Train Loss: 0.1772, Val Loss: 0.1652, Val Accuracy: 94.90%, Val
AUROC: 0.7677, Partial AUROC: 0.0513
Epoch 7/20

Training Epoch 7: 100%|     | 33/33 [01:56<00:00,  3.54s/it,
train_loss=0.158]
Validating Epoch 7: 100%|     | 24/24 [01:24<00:00,  3.54s/it,
val_loss=0.29]

Epoch [6/20], Train Loss: 0.1582, Val Loss: 0.2899, Val Accuracy: 86.44%, Val
AUROC: 0.8396, Partial AUROC: 0.0951
Epoch 8/20

Training Epoch 8: 100%|     | 33/33 [01:45<00:00,  3.21s/it,
train_loss=0.136]
Validating Epoch 8: 100%|     | 24/24 [01:41<00:00,  4.24s/it,
val_loss=0.628]

Epoch [7/20], Train Loss: 0.1359, Val Loss: 0.6281, Val Accuracy: 66.71%, Val
AUROC: 0.8237, Partial AUROC: 0.0860
Epoch 9/20

Training Epoch 9: 100%|     | 33/33 [01:45<00:00,  3.20s/it,
train_loss=0.104]
Validating Epoch 9: 100%|     | 24/24 [01:37<00:00,  4.04s/it,
val_loss=0.2]

Epoch [8/20], Train Loss: 0.1035, Val Loss: 0.2000, Val Accuracy: 92.28%, Val
AUROC: 0.8204, Partial AUROC: 0.0847
Epoch 10/20

Training Epoch 10: 100%|     | 33/33 [01:58<00:00,  3.60s/it,
train_loss=0.118]
Validating Epoch 10: 100%|     | 24/24 [01:25<00:00,  3.55s/it,
val_loss=0.423]

Epoch [9/20], Train Loss: 0.1183, Val Loss: 0.4230, Val Accuracy: 83.96%, Val
AUROC: 0.8264, Partial AUROC: 0.0788
Epoch 11/20

Training Epoch 11: 100%|     | 33/33 [01:45<00:00,  3.19s/it,
train_loss=0.0861]
Validating Epoch 11: 100%|     | 24/24 [01:24<00:00,  3.53s/it,
val_loss=0.547]

Epoch [10/20], Train Loss: 0.0861, Val Loss: 0.5465, Val Accuracy: 83.76%, Val
AUROC: 0.7399, Partial AUROC: 0.0643
Early stopping triggered at epoch 10
Best Epoch: 6, Best Validation Loss: 0.1652
Training Complete
```





Classification Report:

	precision	recall	f1-score	support
Class 0	0.97	0.86	0.91	1431
Class 1	0.10	0.37	0.15	59
accuracy			0.84	1490

```
   macro avg       0.53      0.61      0.53      1490
weighted avg       0.94      0.84      0.88      1490
```

[20]:
```
CustomImageFeatureResNet(
  (resnet): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
    (4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (5): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (6): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
```

```
    )
    (7): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (8): AdaptiveAvgPool2d(output_size=(1, 1))
  )
  (fc_image): Linear(in_features=512, 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)
)
```

5.6 Model 5

[21]:
```
model5 = CustomImageFeatureResNet(feature_input_size=9)  # Assuming 9 features␣
 ↪for metadata
model5.to(device)
# Initialize optimizer
optimizer = optim.SGD(model5.parameters(), lr=0.001)
# Define the loss function with the class weights
criterion = nn.BCELoss()  # Binary classification loss
```

```python
# Set the number of epochs
epochs = 20
best_model_path = "best_model5.pth"
```

```
/home/jupyter-sohka/.local/lib/python3.10/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(
/home/jupyter-sohka/.local/lib/python3.10/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)
```

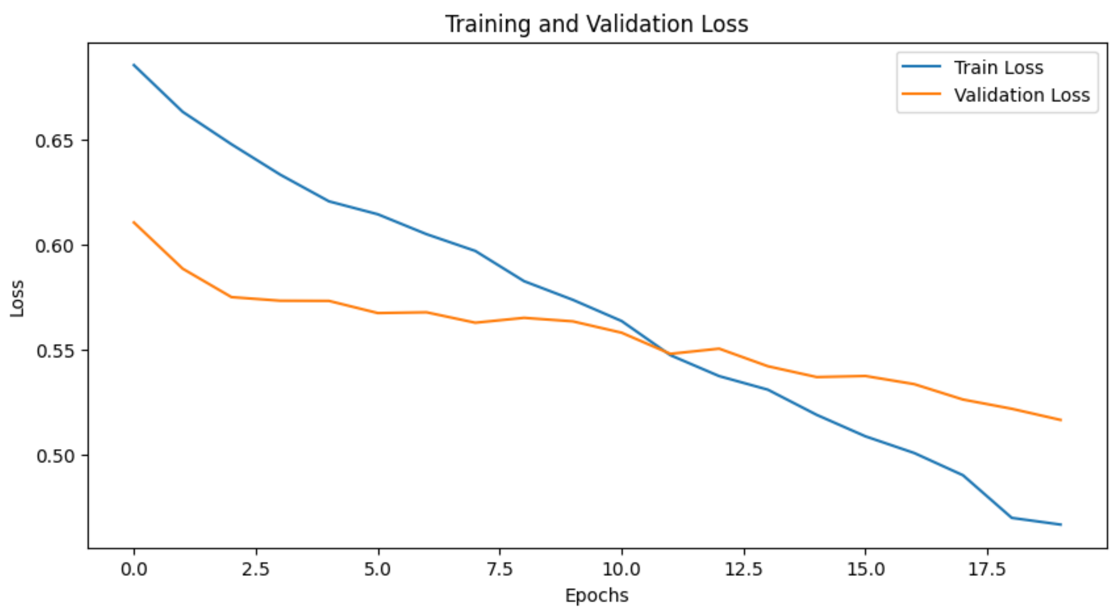
[22]:
```python
train_and_validate(model5,resnet_train_dataloader, resnet_val_dataloader,
    criterion, optimizer, epochs, device, best_model_path )
```

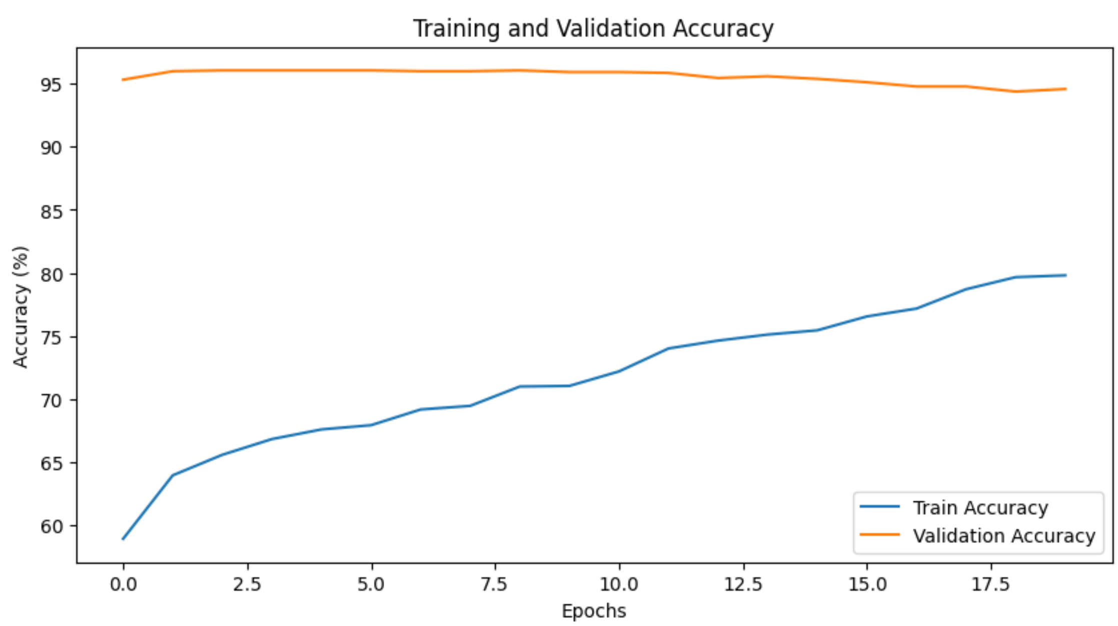
```
Epoch 1/20

Training Epoch 1: 100%|     | 33/33 [01:56<00:00,  3.53s/it,
train_loss=0.685]
Validating Epoch 1: 100%|     | 24/24 [01:32<00:00,  3.84s/it,
val_loss=0.611]

Epoch [0/20], Train Loss: 0.6855, Val Loss: 0.6106, Val Accuracy: 95.30%, Val
AUROC: 0.4326, Partial AUROC: 0.0074
Epoch 2/20

Training Epoch 2: 100%|     | 33/33 [01:57<00:00,  3.56s/it,
train_loss=0.663]
Validating Epoch 2: 100%|     | 24/24 [01:26<00:00,  3.62s/it,
val_loss=0.589]

Epoch [1/20], Train Loss: 0.6632, Val Loss: 0.5886, Val Accuracy: 95.97%, Val
AUROC: 0.4588, Partial AUROC: 0.0115
Epoch 3/20

Training Epoch 3: 100%|     | 33/33 [01:46<00:00,  3.21s/it,
train_loss=0.648]
Validating Epoch 3: 100%|     | 24/24 [01:37<00:00,  4.05s/it,
val_loss=0.575]

Epoch [2/20], Train Loss: 0.6478, Val Loss: 0.5750, Val Accuracy: 96.04%, Val
AUROC: 0.4947, Partial AUROC: 0.0135
Epoch 4/20

Training Epoch 4: 100%|     | 33/33 [01:45<00:00,  3.21s/it,
train_loss=0.633]
```

```
Validating Epoch 4: 100%|     | 24/24 [01:34<00:00,  3.95s/it,
val_loss=0.573]

Epoch [3/20], Train Loss: 0.6333, Val Loss: 0.5733, Val Accuracy: 96.04%, Val
AUROC: 0.5413, Partial AUROC: 0.0237
Epoch 5/20

Training Epoch 5: 100%|     | 33/33 [01:57<00:00,  3.56s/it,
train_loss=0.621]
Validating Epoch 5: 100%|     | 24/24 [01:25<00:00,  3.55s/it,
val_loss=0.573]

Epoch [4/20], Train Loss: 0.6206, Val Loss: 0.5732, Val Accuracy: 96.04%, Val
AUROC: 0.5904, Partial AUROC: 0.0305
Epoch 6/20

Training Epoch 6: 100%|     | 33/33 [01:52<00:00,  3.42s/it,
train_loss=0.614]
Validating Epoch 6: 100%|     | 24/24 [01:31<00:00,  3.79s/it,
val_loss=0.567]

Epoch [5/20], Train Loss: 0.6145, Val Loss: 0.5674, Val Accuracy: 96.04%, Val
AUROC: 0.6290, Partial AUROC: 0.0342
Epoch 7/20

Training Epoch 7: 100%|     | 33/33 [01:50<00:00,  3.34s/it,
train_loss=0.605]
Validating Epoch 7: 100%|     | 24/24 [01:39<00:00,  4.16s/it,
val_loss=0.568]

Epoch [6/20], Train Loss: 0.6050, Val Loss: 0.5678, Val Accuracy: 95.97%, Val
AUROC: 0.6706, Partial AUROC: 0.0446
Epoch 8/20

Training Epoch 8: 100%|     | 33/33 [01:45<00:00,  3.19s/it,
train_loss=0.597]
Validating Epoch 8: 100%|     | 24/24 [01:22<00:00,  3.44s/it,
val_loss=0.563]

Epoch [7/20], Train Loss: 0.5970, Val Loss: 0.5628, Val Accuracy: 95.97%, Val
AUROC: 0.6933, Partial AUROC: 0.0447
Epoch 9/20

Training Epoch 9: 100%|     | 33/33 [01:51<00:00,  3.37s/it,
train_loss=0.583]
Validating Epoch 9: 100%|     | 24/24 [01:26<00:00,  3.61s/it,
val_loss=0.565]

Epoch [8/20], Train Loss: 0.5826, Val Loss: 0.5652, Val Accuracy: 96.04%, Val
AUROC: 0.7087, Partial AUROC: 0.0484
Epoch 10/20

Training Epoch 10: 100%|     | 33/33 [01:44<00:00,  3.18s/it,
train_loss=0.574]
```

```
Validating Epoch 10: 100%|     | 24/24 [01:42<00:00,  4.29s/it,
val_loss=0.563]

Epoch [9/20], Train Loss: 0.5737, Val Loss: 0.5635, Val Accuracy: 95.91%, Val
AUROC: 0.7322, Partial AUROC: 0.0526
Epoch 11/20

Training Epoch 11: 100%|     | 33/33 [01:44<00:00,  3.17s/it,
train_loss=0.564]
Validating Epoch 11: 100%|     | 24/24 [01:28<00:00,  3.70s/it,
val_loss=0.558]

Epoch [10/20], Train Loss: 0.5636, Val Loss: 0.5581, Val Accuracy: 95.91%, Val
AUROC: 0.7473, Partial AUROC: 0.0563
Epoch 12/20

Training Epoch 12: 100%|     | 33/33 [01:43<00:00,  3.13s/it,
train_loss=0.547]
Validating Epoch 12: 100%|     | 24/24 [01:28<00:00,  3.68s/it,
val_loss=0.548]

Epoch [11/20], Train Loss: 0.5474, Val Loss: 0.5480, Val Accuracy: 95.84%, Val
AUROC: 0.7580, Partial AUROC: 0.0597
Epoch 13/20

Training Epoch 13: 100%|     | 33/33 [02:01<00:00,  3.67s/it,
train_loss=0.537]
Validating Epoch 13: 100%|     | 24/24 [01:27<00:00,  3.65s/it,
val_loss=0.55]

Epoch [12/20], Train Loss: 0.5374, Val Loss: 0.5505, Val Accuracy: 95.44%, Val
AUROC: 0.7698, Partial AUROC: 0.0624
Epoch 14/20

Training Epoch 14: 100%|     | 33/33 [01:57<00:00,  3.55s/it,
train_loss=0.531]
Validating Epoch 14: 100%|     | 24/24 [01:25<00:00,  3.55s/it,
val_loss=0.542]

Epoch [13/20], Train Loss: 0.5309, Val Loss: 0.5421, Val Accuracy: 95.57%, Val
AUROC: 0.7771, Partial AUROC: 0.0634
Epoch 15/20

Training Epoch 15: 100%|     | 33/33 [01:44<00:00,  3.18s/it,
train_loss=0.519]
Validating Epoch 15: 100%|     | 24/24 [01:35<00:00,  3.98s/it,
val_loss=0.537]

Epoch [14/20], Train Loss: 0.5190, Val Loss: 0.5370, Val Accuracy: 95.37%, Val
AUROC: 0.7810, Partial AUROC: 0.0652
Epoch 16/20

Training Epoch 16: 100%|     | 33/33 [01:47<00:00,  3.27s/it,
train_loss=0.509]
```

```
Validating Epoch 16: 100%|     | 24/24 [01:27<00:00,  3.66s/it,
val_loss=0.537]

Epoch [15/20], Train Loss: 0.5087, Val Loss: 0.5375, Val Accuracy: 95.10%, Val
AUROC: 0.7909, Partial AUROC: 0.0692
Epoch 17/20

Training Epoch 17: 100%|     | 33/33 [01:55<00:00,  3.48s/it,
train_loss=0.501]
Validating Epoch 17: 100%|     | 24/24 [01:22<00:00,  3.44s/it,
val_loss=0.534]

Epoch [16/20], Train Loss: 0.5008, Val Loss: 0.5336, Val Accuracy: 94.77%, Val
AUROC: 0.7943, Partial AUROC: 0.0707
Epoch 18/20

Training Epoch 18: 100%|     | 33/33 [01:44<00:00,  3.17s/it,
train_loss=0.49]
Validating Epoch 18: 100%|     | 24/24 [01:34<00:00,  3.93s/it,
val_loss=0.526]

Epoch [17/20], Train Loss: 0.4902, Val Loss: 0.5263, Val Accuracy: 94.77%, Val
AUROC: 0.7982, Partial AUROC: 0.0720
Epoch 19/20

Training Epoch 19: 100%|     | 33/33 [01:44<00:00,  3.17s/it,
train_loss=0.47]
Validating Epoch 19: 100%|     | 24/24 [01:33<00:00,  3.91s/it,
val_loss=0.522]

Epoch [18/20], Train Loss: 0.4699, Val Loss: 0.5219, Val Accuracy: 94.36%, Val
AUROC: 0.8041, Partial AUROC: 0.0740
Epoch 20/20

Training Epoch 20: 100%|     | 33/33 [01:54<00:00,  3.47s/it,
train_loss=0.467]
Validating Epoch 20: 100%|     | 24/24 [01:23<00:00,  3.48s/it,
val_loss=0.517]

Epoch [19/20], Train Loss: 0.4667, Val Loss: 0.5166, Val Accuracy: 94.56%, Val
AUROC: 0.8086, Partial AUROC: 0.0749
Best Epoch: 20, Best Validation Loss: 0.5166
Training Complete
```






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

     Class 0       0.97      0.97      0.97      1431
     Class 1       0.30      0.27      0.28        59

    accuracy                           0.95      1490
```

```
   macro avg       0.63      0.62      0.63      1490
weighted avg       0.94      0.95      0.94      1490
```

[22]:
```
CustomImageFeatureResNet(
  (resnet): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
    (4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (5): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (6): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
```

```
    )
    (7): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (8): AdaptiveAvgPool2d(output_size=(1, 1))
  )
  (fc_image): Linear(in_features=512, 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)
)
```

5.7 Model 6

[23]:
```
model6 = CustomImageFeatureResNet(feature_input_size=9)  # Assuming 9 features
 ↪for metadata
model6.to(device)
# Initialize optimizer
optimizer = optim.SGD(model6.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
best_model_path = "best_model6.pth"
```

```
/home/jupyter-sohka/.local/lib/python3.10/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(
/home/jupyter-sohka/.local/lib/python3.10/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)
```

[24]:
```
resnet_train_dataloader = DataLoader(resnet_train_dataset,␣
 ↪batch_size=batch_size, shuffle=True)
resnet_val_dataloader = DataLoader(resnet_val_dataset, batch_size=batch_size,␣
 ↪shuffle=True)
```

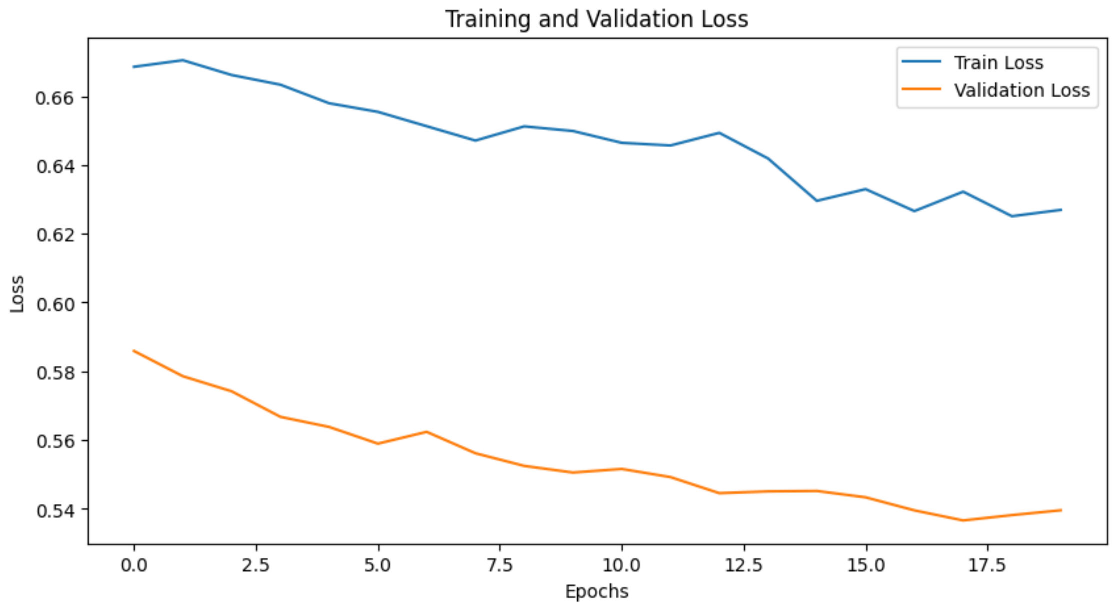
[25]:
```
train_and_validate(model6,resnet_train_dataloader, resnet_val_dataloader,␣
 ↪criterion, optimizer, epochs, device, best_model_path )
```

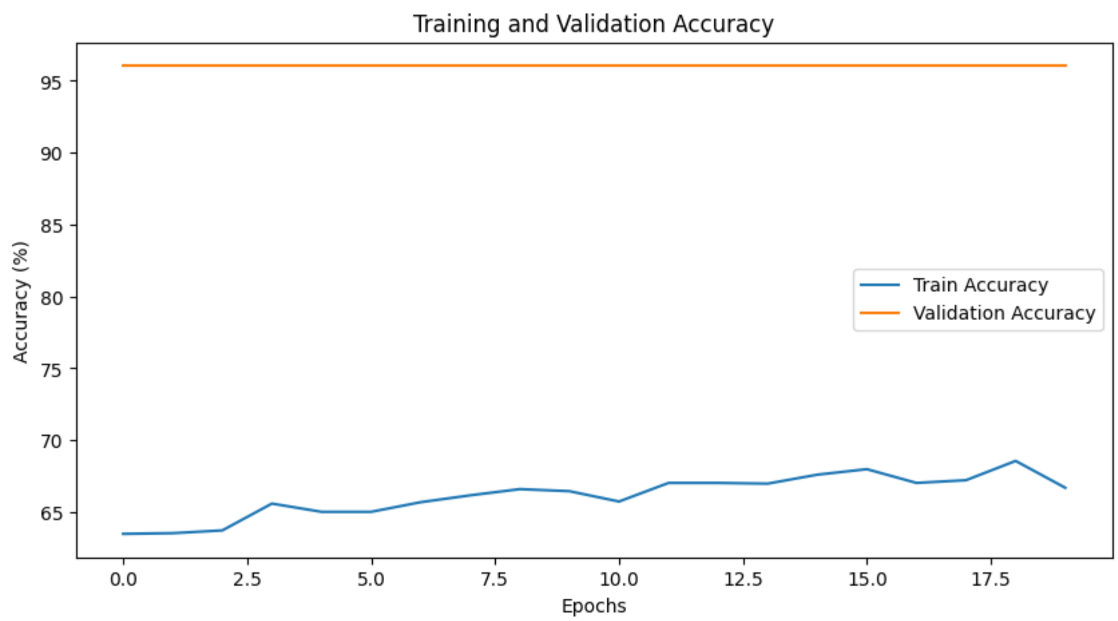
```
Epoch 1/20

Training Epoch 1: 100%|     | 66/66 [01:41<00:00,  1.53s/it,
train_loss=0.669]
Validating Epoch 1: 100%|     | 47/47 [01:29<00:00,  1.90s/it,
val_loss=0.586]

Epoch [0/20], Train Loss: 0.6685, Val Loss: 0.5859, Val Accuracy: 96.04%, Val
AUROC: 0.4509, Partial AUROC: 0.0088
Epoch 2/20

Training Epoch 2: 100%|     | 66/66 [01:36<00:00,  1.47s/it,
train_loss=0.67]
Validating Epoch 2: 100%|     | 47/47 [01:20<00:00,  1.71s/it,
val_loss=0.579]

Epoch [1/20], Train Loss: 0.6704, Val Loss: 0.5786, Val Accuracy: 96.04%, Val
AUROC: 0.4554, Partial AUROC: 0.0107
Epoch 3/20

Training Epoch 3: 100%|     | 66/66 [01:50<00:00,  1.67s/it,
train_loss=0.666]
Validating Epoch 3: 100%|     | 47/47 [01:14<00:00,  1.59s/it,
val_loss=0.574]
```

```
Epoch [2/20], Train Loss: 0.6661, Val Loss: 0.5742, Val Accuracy: 96.04%, Val
AUROC: 0.4600, Partial AUROC: 0.0093
Epoch 4/20

Training Epoch 4: 100%|     | 66/66 [01:36<00:00,  1.47s/it,
train_loss=0.663]
Validating Epoch 4: 100%|     | 47/47 [01:13<00:00,  1.57s/it,
val_loss=0.567]

Epoch [3/20], Train Loss: 0.6633, Val Loss: 0.5668, Val Accuracy: 96.04%, Val
AUROC: 0.4650, Partial AUROC: 0.0108
Epoch 5/20

Training Epoch 5: 100%|     | 66/66 [01:49<00:00,  1.66s/it,
train_loss=0.658]
Validating Epoch 5: 100%|     | 47/47 [01:14<00:00,  1.58s/it,
val_loss=0.564]

Epoch [4/20], Train Loss: 0.6579, Val Loss: 0.5638, Val Accuracy: 96.04%, Val
AUROC: 0.4679, Partial AUROC: 0.0114
Epoch 6/20

Training Epoch 6: 100%|     | 66/66 [01:43<00:00,  1.58s/it,
train_loss=0.655]
Validating Epoch 6: 100%|     | 47/47 [01:19<00:00,  1.70s/it,
val_loss=0.559]

Epoch [5/20], Train Loss: 0.6554, Val Loss: 0.5589, Val Accuracy: 96.04%, Val
AUROC: 0.4788, Partial AUROC: 0.0118
Epoch 7/20

Training Epoch 7: 100%|     | 66/66 [01:47<00:00,  1.63s/it,
train_loss=0.651]
Validating Epoch 7: 100%|     | 47/47 [01:13<00:00,  1.55s/it,
val_loss=0.562]

Epoch [6/20], Train Loss: 0.6513, Val Loss: 0.5624, Val Accuracy: 96.04%, Val
AUROC: 0.4857, Partial AUROC: 0.0125
Epoch 8/20

Training Epoch 8: 100%|     | 66/66 [01:35<00:00,  1.44s/it,
train_loss=0.647]
Validating Epoch 8: 100%|     | 47/47 [01:25<00:00,  1.82s/it,
val_loss=0.556]

Epoch [7/20], Train Loss: 0.6471, Val Loss: 0.5562, Val Accuracy: 96.04%, Val
AUROC: 0.4861, Partial AUROC: 0.0150
Epoch 9/20

Training Epoch 9: 100%|     | 66/66 [01:44<00:00,  1.58s/it,
train_loss=0.651]
Validating Epoch 9: 100%|     | 47/47 [01:13<00:00,  1.57s/it,
val_loss=0.552]
```

```
Epoch [8/20], Train Loss: 0.6512, Val Loss: 0.5525, Val Accuracy: 96.04%, Val
AUROC: 0.4862, Partial AUROC: 0.0144
Epoch 10/20

Training Epoch 10: 100%|     | 66/66 [01:46<00:00,  1.61s/it,
train_loss=0.65]
Validating Epoch 10: 100%|     | 47/47 [01:13<00:00,  1.56s/it,
val_loss=0.551]

Epoch [9/20], Train Loss: 0.6499, Val Loss: 0.5506, Val Accuracy: 96.04%, Val
AUROC: 0.5049, Partial AUROC: 0.0171
Epoch 11/20

Training Epoch 11: 100%|     | 66/66 [01:33<00:00,  1.42s/it,
train_loss=0.646]
Validating Epoch 11: 100%|     | 47/47 [01:13<00:00,  1.56s/it,
val_loss=0.552]

Epoch [10/20], Train Loss: 0.6464, Val Loss: 0.5516, Val Accuracy: 96.04%, Val
AUROC: 0.5077, Partial AUROC: 0.0188
Epoch 12/20

Training Epoch 12: 100%|     | 66/66 [01:50<00:00,  1.67s/it,
train_loss=0.646]
Validating Epoch 12: 100%|     | 47/47 [01:23<00:00,  1.79s/it,
val_loss=0.549]

Epoch [11/20], Train Loss: 0.6457, Val Loss: 0.5492, Val Accuracy: 96.04%, Val
AUROC: 0.5144, Partial AUROC: 0.0179
Epoch 13/20

Training Epoch 13: 100%|     | 66/66 [01:35<00:00,  1.45s/it,
train_loss=0.649]
Validating Epoch 13: 100%|     | 47/47 [01:13<00:00,  1.56s/it,
val_loss=0.545]

Epoch [12/20], Train Loss: 0.6493, Val Loss: 0.5446, Val Accuracy: 96.04%, Val
AUROC: 0.5160, Partial AUROC: 0.0198
Epoch 14/20

Training Epoch 14: 100%|     | 66/66 [01:51<00:00,  1.69s/it,
train_loss=0.642]
Validating Epoch 14: 100%|     | 47/47 [01:14<00:00,  1.58s/it,
val_loss=0.545]

Epoch [13/20], Train Loss: 0.6419, Val Loss: 0.5451, Val Accuracy: 96.04%, Val
AUROC: 0.5254, Partial AUROC: 0.0180
Epoch 15/20

Training Epoch 15: 100%|     | 66/66 [01:34<00:00,  1.43s/it,
train_loss=0.63]
Validating Epoch 15: 100%|     | 47/47 [01:14<00:00,  1.59s/it,
val_loss=0.545]
```

```
Epoch [14/20], Train Loss: 0.6296, Val Loss: 0.5452, Val Accuracy: 96.04%, Val
AUROC: 0.5350, Partial AUROC: 0.0210
Epoch 16/20

Training Epoch 16: 100%|     | 66/66 [01:57<00:00,  1.78s/it,
train_loss=0.633]
Validating Epoch 16: 100%|     | 47/47 [01:15<00:00,  1.60s/it,
val_loss=0.543]

Epoch [15/20], Train Loss: 0.6330, Val Loss: 0.5433, Val Accuracy: 96.04%, Val
AUROC: 0.5368, Partial AUROC: 0.0218
Epoch 17/20

Training Epoch 17: 100%|     | 66/66 [01:35<00:00,  1.45s/it,
train_loss=0.627]
Validating Epoch 17: 100%|     | 47/47 [01:15<00:00,  1.60s/it,
val_loss=0.54]

Epoch [16/20], Train Loss: 0.6266, Val Loss: 0.5396, Val Accuracy: 96.04%, Val
AUROC: 0.5539, Partial AUROC: 0.0234
Epoch 18/20

Training Epoch 18: 100%|     | 66/66 [01:50<00:00,  1.67s/it,
train_loss=0.632]
Validating Epoch 18: 100%|     | 47/47 [01:14<00:00,  1.59s/it,
val_loss=0.537]

Epoch [17/20], Train Loss: 0.6322, Val Loss: 0.5366, Val Accuracy: 96.04%, Val
AUROC: 0.5619, Partial AUROC: 0.0220
Epoch 19/20

Training Epoch 19: 100%|     | 66/66 [01:45<00:00,  1.60s/it,
train_loss=0.625]
Validating Epoch 19: 100%|     | 47/47 [01:15<00:00,  1.60s/it,
val_loss=0.538]

Epoch [18/20], Train Loss: 0.6251, Val Loss: 0.5382, Val Accuracy: 96.04%, Val
AUROC: 0.5609, Partial AUROC: 0.0234
Epoch 20/20

Training Epoch 20: 100%|     | 66/66 [01:48<00:00,  1.64s/it,
train_loss=0.627]
Validating Epoch 20: 100%|     | 47/47 [01:13<00:00,  1.57s/it,
val_loss=0.54]

Epoch [19/20], Train Loss: 0.6269, Val Loss: 0.5396, Val Accuracy: 96.04%, Val
AUROC: 0.5605, Partial AUROC: 0.0246
Best Epoch: 18, Best Validation Loss: 0.5366
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))
```

[25]:
```
CustomImageFeatureResNet(
  (resnet): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
    (4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
```

```
1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (5): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (6): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
```

```
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (7): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (8): AdaptiveAvgPool2d(output_size=(1, 1))
  )
```

```
  (fc_image): Linear(in_features=512, 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)
)
```

5.8 Model 7

[26]:
```python
model7 =  CustomImageFeatureEfficientNet(feature_input_size=9)  # Assuming 9␣
 ↪features for metadata
model7.to(device)
# Initialize optimizer
optimizer = optim.Adam(model7.parameters(), lr= 1.1621608010269284e-05)
# Define the loss function with the class weights
criterion = nn.BCELoss()  # Binary classification loss
# Set the number of epochs
epochs = 20
batch_size = 16
best_model_path = "best_model7.pth"
```

```
/home/jupyter-sohka/.local/lib/python3.10/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(
/home/jupyter-sohka/.local/lib/python3.10/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=EfficientNet_B0_Weights.IMAGENET1K_V1`. You can also use
`weights=EfficientNet_B0_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

[27]:
```python
effnet_train_dataloader = DataLoader(effnet_train_dataset, ␣
 ↪batch_size=batch_size, shuffle=True)
effnet_val_dataloader = DataLoader(effnet_val_dataset,  batch_size=batch_size,␣
 ↪shuffle=True)
```

[28]:
```python
train_and_validate(model7,effnet_train_dataloader, effnet_val_dataloader,␣
 ↪criterion, optimizer, epochs, device, best_model_path )
```

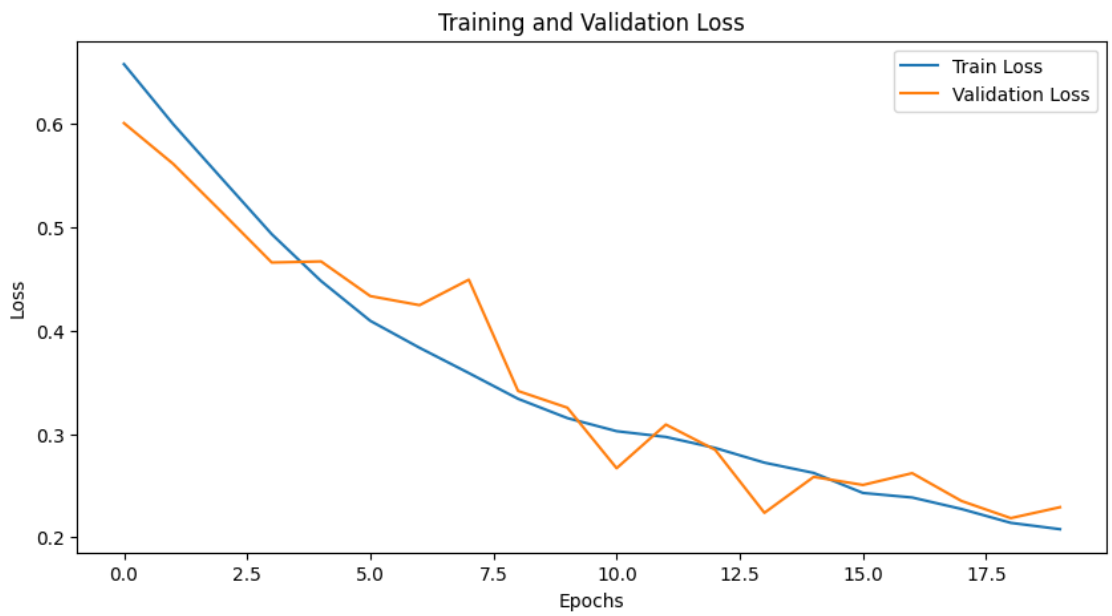
```
Epoch 1/20

Training Epoch 1: 100%|     | 131/131 [01:37<00:00,  1.35it/s,
train_loss=0.657]
Validating Epoch 1: 100%|     | 94/94 [01:16<00:00,  1.23it/s,
val_loss=0.6]
```

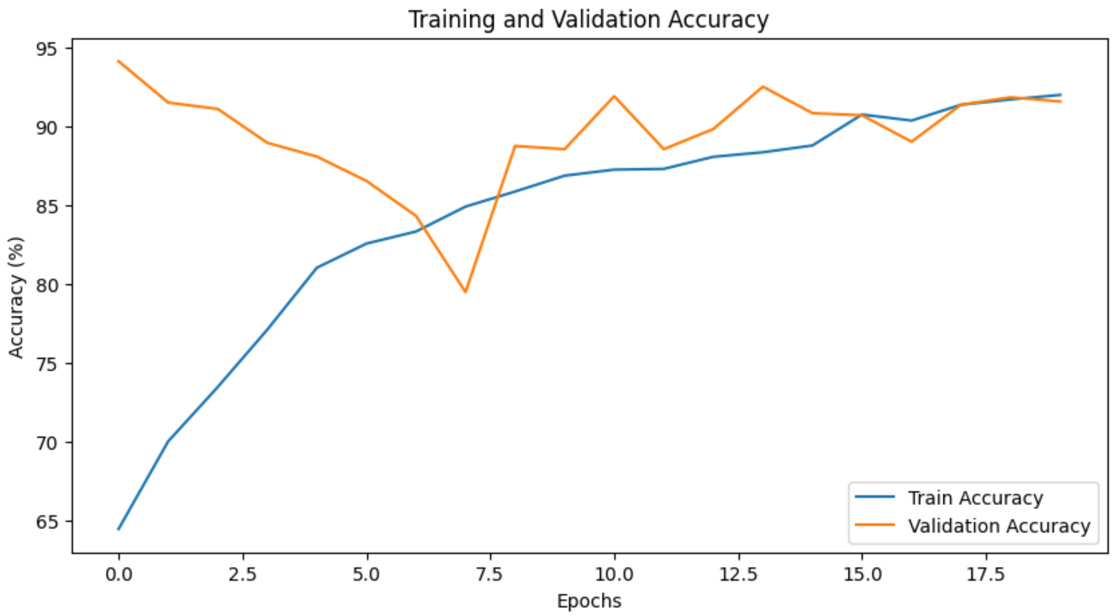
```
Epoch [0/20], Train Loss: 0.6573, Val Loss: 0.6004, Val Accuracy: 94.09%, Val
AUROC: 0.6837, Partial AUROC: 0.0487
Epoch 2/20

Training Epoch 2: 100%|     | 131/131 [01:37<00:00,  1.35it/s,
train_loss=0.599]
Validating Epoch 2: 100%|     | 94/94 [01:14<00:00,  1.26it/s,
val_loss=0.561]

Epoch [1/20], Train Loss: 0.5993, Val Loss: 0.5610, Val Accuracy: 91.48%, Val
AUROC: 0.7223, Partial AUROC: 0.0711
Epoch 3/20

Training Epoch 3: 100%|     | 131/131 [01:51<00:00,  1.18it/s,
train_loss=0.546]
Validating Epoch 3: 100%|     | 94/94 [01:07<00:00,  1.39it/s,
val_loss=0.514]

Epoch [2/20], Train Loss: 0.5460, Val Loss: 0.5136, Val Accuracy: 91.07%, Val
AUROC: 0.7681, Partial AUROC: 0.0788
Epoch 4/20

Training Epoch 4: 100%|     | 131/131 [01:35<00:00,  1.37it/s,
train_loss=0.493]
Validating Epoch 4: 100%|     | 94/94 [01:07<00:00,  1.40it/s,
val_loss=0.466]

Epoch [3/20], Train Loss: 0.4930, Val Loss: 0.4657, Val Accuracy: 88.93%, Val
AUROC: 0.7839, Partial AUROC: 0.0812
Epoch 5/20

Training Epoch 5: 100%|     | 131/131 [01:36<00:00,  1.36it/s,
train_loss=0.448]
Validating Epoch 5: 100%|     | 94/94 [01:20<00:00,  1.17it/s,
val_loss=0.467]

Epoch [4/20], Train Loss: 0.4478, Val Loss: 0.4667, Val Accuracy: 88.05%, Val
AUROC: 0.8297, Partial AUROC: 0.0929
Epoch 6/20

Training Epoch 6: 100%|     | 131/131 [01:46<00:00,  1.23it/s,
train_loss=0.409]
Validating Epoch 6: 100%|     | 94/94 [01:10<00:00,  1.33it/s,
val_loss=0.433]

Epoch [5/20], Train Loss: 0.4093, Val Loss: 0.4332, Val Accuracy: 86.51%, Val
AUROC: 0.8174, Partial AUROC: 0.0877
Epoch 7/20

Training Epoch 7: 100%|     | 131/131 [01:42<00:00,  1.28it/s,
train_loss=0.383]
Validating Epoch 7: 100%|     | 94/94 [01:22<00:00,  1.14it/s,
val_loss=0.424]
```

```
Epoch [6/20], Train Loss: 0.3832, Val Loss: 0.4244, Val Accuracy: 84.30%, Val
AUROC: 0.8197, Partial AUROC: 0.0877
Epoch 8/20

Training Epoch 8: 100%|     | 131/131 [01:41<00:00,  1.29it/s,
train_loss=0.359]
Validating Epoch 8: 100%|     | 94/94 [01:12<00:00,  1.29it/s,
val_loss=0.449]

Epoch [7/20], Train Loss: 0.3588, Val Loss: 0.4490, Val Accuracy: 79.46%, Val
AUROC: 0.8550, Partial AUROC: 0.0983
Epoch 9/20

Training Epoch 9: 100%|     | 131/131 [02:00<00:00,  1.09it/s,
train_loss=0.334]
Validating Epoch 9: 100%|     | 94/94 [01:11<00:00,  1.31it/s,
val_loss=0.341]

Epoch [8/20], Train Loss: 0.3339, Val Loss: 0.3415, Val Accuracy: 88.72%, Val
AUROC: 0.8417, Partial AUROC: 0.0889
Epoch 10/20

Training Epoch 10: 100%|     | 131/131 [01:41<00:00,  1.29it/s,
train_loss=0.315]
Validating Epoch 10: 100%|     | 94/94 [01:11<00:00,  1.32it/s,
val_loss=0.325]

Epoch [9/20], Train Loss: 0.3153, Val Loss: 0.3253, Val Accuracy: 88.52%, Val
AUROC: 0.8501, Partial AUROC: 0.0920
Epoch 11/20

Training Epoch 11: 100%|     | 131/131 [01:54<00:00,  1.15it/s,
train_loss=0.303]
Validating Epoch 11: 100%|     | 94/94 [01:11<00:00,  1.31it/s,
val_loss=0.267]

Epoch [10/20], Train Loss: 0.3026, Val Loss: 0.2668, Val Accuracy: 91.88%, Val
AUROC: 0.8576, Partial AUROC: 0.0994
Epoch 12/20

Training Epoch 12: 100%|     | 131/131 [01:43<00:00,  1.26it/s,
train_loss=0.297]
Validating Epoch 12: 100%|     | 94/94 [01:27<00:00,  1.07it/s,
val_loss=0.309]

Epoch [11/20], Train Loss: 0.2971, Val Loss: 0.3090, Val Accuracy: 88.52%, Val
AUROC: 0.8616, Partial AUROC: 0.1016
Epoch 13/20

Training Epoch 13: 100%|     | 131/131 [01:42<00:00,  1.28it/s,
train_loss=0.286]
Validating Epoch 13: 100%|     | 94/94 [01:11<00:00,  1.31it/s,
val_loss=0.285]
```

```
Epoch [12/20], Train Loss: 0.2864, Val Loss: 0.2846, Val Accuracy: 89.80%, Val
AUROC: 0.8568, Partial AUROC: 0.1048
Epoch 14/20

Training Epoch 14: 100%|     | 131/131 [01:42<00:00,  1.28it/s,
train_loss=0.272]
Validating Epoch 14: 100%|     | 94/94 [01:13<00:00,  1.28it/s,
val_loss=0.224]

Epoch [13/20], Train Loss: 0.2722, Val Loss: 0.2237, Val Accuracy: 92.48%, Val
AUROC: 0.8562, Partial AUROC: 0.0960
Epoch 15/20

Training Epoch 15: 100%|     | 131/131 [01:54<00:00,  1.14it/s,
train_loss=0.262]
Validating Epoch 15: 100%|     | 94/94 [01:12<00:00,  1.30it/s,
val_loss=0.258]

Epoch [14/20], Train Loss: 0.2623, Val Loss: 0.2585, Val Accuracy: 90.81%, Val
AUROC: 0.8685, Partial AUROC: 0.1043
Epoch 16/20

Training Epoch 16: 100%|     | 131/131 [01:48<00:00,  1.21it/s,
train_loss=0.243]
Validating Epoch 16: 100%|     | 94/94 [01:25<00:00,  1.10it/s,
val_loss=0.251]

Epoch [15/20], Train Loss: 0.2429, Val Loss: 0.2507, Val Accuracy: 90.67%, Val
AUROC: 0.8594, Partial AUROC: 0.0991
Epoch 17/20

Training Epoch 17: 100%|     | 131/131 [01:42<00:00,  1.28it/s,
train_loss=0.239]
Validating Epoch 17: 100%|     | 94/94 [01:09<00:00,  1.34it/s,
val_loss=0.262]

Epoch [16/20], Train Loss: 0.2385, Val Loss: 0.2621, Val Accuracy: 88.99%, Val
AUROC: 0.8689, Partial AUROC: 0.1054
Epoch 18/20

Training Epoch 18: 100%|     | 131/131 [01:38<00:00,  1.33it/s,
train_loss=0.227]
Validating Epoch 18: 100%|     | 94/94 [01:09<00:00,  1.36it/s,
val_loss=0.235]

Epoch [17/20], Train Loss: 0.2274, Val Loss: 0.2352, Val Accuracy: 91.34%, Val
AUROC: 0.8629, Partial AUROC: 0.0992
Epoch 19/20

Training Epoch 19: 100%|     | 131/131 [01:58<00:00,  1.11it/s,
train_loss=0.214]
Validating Epoch 19: 100%|     | 94/94 [01:09<00:00,  1.36it/s,
val_loss=0.219]
```

```
Epoch [18/20], Train Loss: 0.2141, Val Loss: 0.2186, Val Accuracy: 91.81%, Val
AUROC: 0.8671, Partial AUROC: 0.1039
Epoch 20/20

Training Epoch 20: 100%|      | 131/131 [01:40<00:00,  1.30it/s,
train_loss=0.208]
Validating Epoch 20: 100%|      | 94/94 [01:08<00:00,  1.37it/s,
val_loss=0.229]

Epoch [19/20], Train Loss: 0.2079, Val Loss: 0.2292, Val Accuracy: 91.54%, Val
AUROC: 0.8535, Partial AUROC: 0.0964
Best Epoch: 19, Best Validation Loss: 0.2186
Training Complete
```






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

     Class 0       0.98      0.93      0.95      1431
     Class 1       0.26      0.61      0.36        59

    accuracy                           0.92      1490
   macro avg       0.62      0.77      0.66      1490
weighted avg       0.95      0.92      0.93      1490
```

[28]:
```
CustomImageFeatureEfficientNet(
  (efficientnet): Sequential(
    (0): Sequential(
      (0): Conv2dNormActivation(
        (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): SiLU(inplace=True)
      )
      (1): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), groups=32, bias=False)
```

```
              (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(32, 8, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(8, 32, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (2): Conv2dNormActivation(
              (0): Conv2d(32, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.0, mode=row)
        )
      )
      (2): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(16, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(96, 96, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), groups=96, bias=False)
              (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(96, 4, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(4, 96, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(96, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (1): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
            )
          )
          (stochastic_depth): StochasticDepth(p=0.0125, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(24, 144, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(144, 144, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=144, bias=False)
              (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(144, 6, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(6, 144, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(144, 24, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.025, mode=row)
        )
      )
      (3): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(24, 144, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
```

```
            (1): Conv2dNormActivation(
              (0): Conv2d(144, 144, kernel_size=(5, 5), stride=(2, 2),
padding=(2, 2), groups=144, bias=False)
              (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(144, 6, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(6, 144, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(144, 40, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(40, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.037500000000000006, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(40, 240, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(240, 240, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=240, bias=False)
              (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(240, 10, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(10, 240, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
```

```
              (0): Conv2d(240, 40, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(40, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.05, mode=row)
        )
      )
      (4): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(40, 240, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(240, 240, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=240, bias=False)
              (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(240, 10, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(10, 240, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(240, 80, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.0625, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(80, 480, kernel_size=(1, 1), stride=(1, 1),
bias=False)
```

```
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(480, 480, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=480, bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(480, 20, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(20, 480, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(480, 80, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.07500000000000001, mode=row)
        )
        (2): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(80, 480, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(480, 480, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=480, bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(480, 20, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(20, 480, kernel_size=(1, 1), stride=(1, 1))
```

```
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(480, 80, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.08750000000000001, mode=row)
      )
    )
    (5): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(80, 480, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(480, 480, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=480, bias=False)
            (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(480, 20, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(20, 480, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(480, 112, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.1, mode=row)
      )
      (1): MBConv(
```

```
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(112, 672, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(672, 672, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=672, bias=False)
              (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(672, 28, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(28, 672, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(672, 112, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.1125, mode=row)
        )
        (2): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(112, 672, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(672, 672, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=672, bias=False)
              (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
```

```
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(672, 28, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(28, 672, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(672, 112, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.125, mode=row)
        )
      )
      (6): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(112, 672, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(672, 672, kernel_size=(5, 5), stride=(2, 2),
padding=(2, 2), groups=672, bias=False)
              (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(672, 28, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(28, 672, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(672, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
```

```
          )
          (stochastic_depth): StochasticDepth(p=0.1375, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(1152, 1152, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=1152, bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(1152, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.15000000000000002, mode=row)
        )
        (2): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(1152, 1152, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=1152, bias=False)
```

```
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(1152, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.1625, mode=row)
        )
        (3): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(1152, 1152, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=1152, bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(1152, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.17500000000000002, mode=row)
        )
      )
      (7): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(1152, 1152, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1152, bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(1152, 320, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(320, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.1875, mode=row)
        )
      )
      (8): Conv2dNormActivation(
        (0): Conv2d(320, 1280, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(1280, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): SiLU(inplace=True)
      )
    )
```

```
    (1): AdaptiveAvgPool2d(output_size=1)
  )
  (fc_image): Linear(in_features=1280, 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)
)
```

5.9 Model 8

[29]:
```python
model8 = CustomImageFeatureEfficientNet(feature_input_size=9)  # Assuming 9␣
↪features for metadata
model8.to(device)
# Initialize optimizer
optimizer = optim.SGD(model8.parameters(), lr=0.01)
# Define the loss function with the class weights
criterion = nn.BCELoss()  # Binary classification loss
# Set the number of epochs
epochs = 20
best_model_path = "best_model8.pth"
```

```
/home/jupyter-sohka/.local/lib/python3.10/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(
/home/jupyter-sohka/.local/lib/python3.10/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=EfficientNet_B0_Weights.IMAGENET1K_V1`. You can also use
`weights=EfficientNet_B0_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

[30]:
```python
train_and_validate(model8,effnet_train_dataloader, effnet_val_dataloader,␣
↪criterion, optimizer, epochs, device, best_model_path )
```

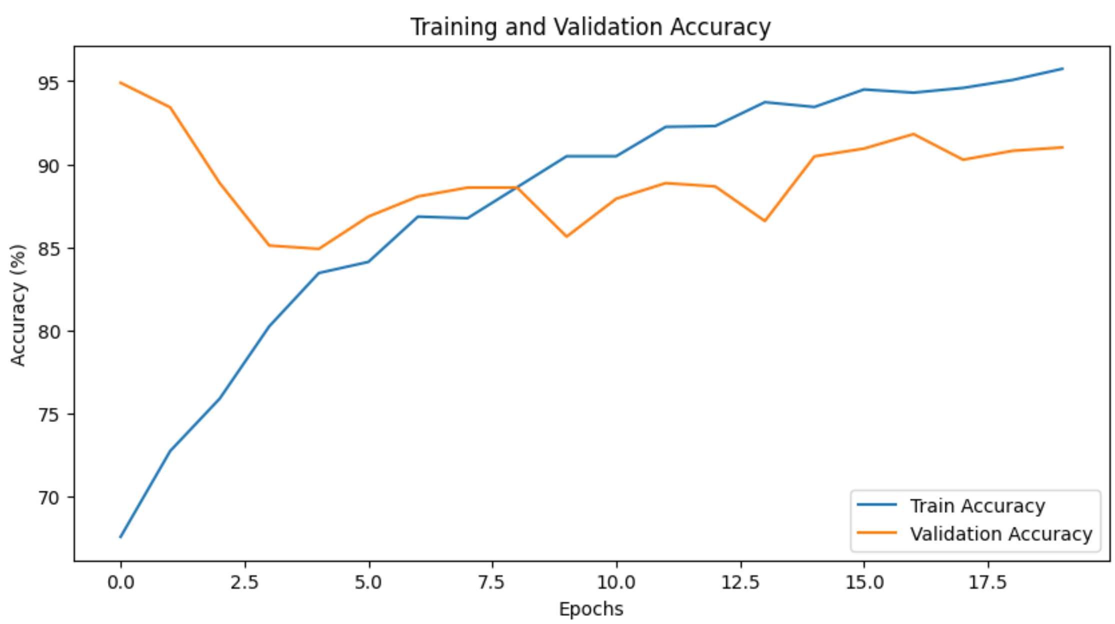
```
Epoch 1/20

Training Epoch 1: 100%|     | 131/131 [01:49<00:00,  1.19it/s,
train_loss=0.624]
Validating Epoch 1: 100%|     | 94/94 [01:09<00:00,  1.35it/s,
val_loss=0.48]

Epoch [0/20], Train Loss: 0.6243, Val Loss: 0.4802, Val Accuracy: 94.90%, Val
AUROC: 0.6897, Partial AUROC: 0.0421
Epoch 2/20
```

```
Training Epoch 2: 100%|     | 131/131 [01:34<00:00,  1.38it/s,
train_loss=0.558]
Validating Epoch 2: 100%|     | 94/94 [01:14<00:00,  1.27it/s,
val_loss=0.435]

Epoch [1/20], Train Loss: 0.5583, Val Loss: 0.4353, Val Accuracy: 93.42%, Val
AUROC: 0.7293, Partial AUROC: 0.0499
Epoch 3/20

Training Epoch 3: 100%|     | 131/131 [01:51<00:00,  1.18it/s,
train_loss=0.508]
Validating Epoch 3: 100%|     | 94/94 [01:09<00:00,  1.36it/s,
val_loss=0.469]

Epoch [2/20], Train Loss: 0.5078, Val Loss: 0.4694, Val Accuracy: 88.86%, Val
AUROC: 0.7578, Partial AUROC: 0.0519
Epoch 4/20

Training Epoch 4: 100%|     | 131/131 [01:40<00:00,  1.31it/s,
train_loss=0.446]
Validating Epoch 4: 100%|     | 94/94 [01:09<00:00,  1.35it/s,
val_loss=0.484]

Epoch [3/20], Train Loss: 0.4456, Val Loss: 0.4837, Val Accuracy: 85.10%, Val
AUROC: 0.8050, Partial AUROC: 0.0673
Epoch 5/20

Training Epoch 5: 100%|     | 131/131 [01:54<00:00,  1.14it/s,
train_loss=0.395]
Validating Epoch 5: 100%|     | 94/94 [01:12<00:00,  1.30it/s,
val_loss=0.446]

Epoch [4/20], Train Loss: 0.3946, Val Loss: 0.4457, Val Accuracy: 84.90%, Val
AUROC: 0.8304, Partial AUROC: 0.0794
Epoch 6/20

Training Epoch 6: 100%|     | 131/131 [01:47<00:00,  1.22it/s,
train_loss=0.356]
Validating Epoch 6: 100%|     | 94/94 [01:11<00:00,  1.32it/s,
val_loss=0.386]

Epoch [5/20], Train Loss: 0.3556, Val Loss: 0.3865, Val Accuracy: 86.85%, Val
AUROC: 0.8519, Partial AUROC: 0.0918
Epoch 7/20

Training Epoch 7: 100%|     | 131/131 [01:54<00:00,  1.14it/s,
train_loss=0.314]
Validating Epoch 7: 100%|     | 94/94 [01:10<00:00,  1.33it/s,
val_loss=0.338]

Epoch [6/20], Train Loss: 0.3138, Val Loss: 0.3384, Val Accuracy: 88.05%, Val
AUROC: 0.8491, Partial AUROC: 0.0907
Epoch 8/20
```

```
Training Epoch 8: 100%|     | 131/131 [01:39<00:00,  1.32it/s,
train_loss=0.303]
Validating Epoch 8: 100%|     | 94/94 [01:10<00:00,  1.34it/s,
val_loss=0.311]

Epoch [7/20], Train Loss: 0.3029, Val Loss: 0.3105, Val Accuracy: 88.59%, Val
AUROC: 0.8665, Partial AUROC: 0.1018
Epoch 9/20

Training Epoch 9: 100%|     | 131/131 [01:51<00:00,  1.18it/s,
train_loss=0.268]
Validating Epoch 9: 100%|     | 94/94 [01:15<00:00,  1.25it/s,
val_loss=0.294]

Epoch [8/20], Train Loss: 0.2681, Val Loss: 0.2938, Val Accuracy: 88.59%, Val
AUROC: 0.8788, Partial AUROC: 0.1111
Epoch 10/20

Training Epoch 10: 100%|     | 131/131 [01:39<00:00,  1.31it/s,
train_loss=0.24]
Validating Epoch 10: 100%|     | 94/94 [01:11<00:00,  1.31it/s,
val_loss=0.331]

Epoch [9/20], Train Loss: 0.2401, Val Loss: 0.3313, Val Accuracy: 85.64%, Val
AUROC: 0.8583, Partial AUROC: 0.1039
Epoch 11/20

Training Epoch 11: 100%|     | 131/131 [01:54<00:00,  1.15it/s,
train_loss=0.229]
Validating Epoch 11: 100%|     | 94/94 [01:12<00:00,  1.29it/s,
val_loss=0.291]

Epoch [10/20], Train Loss: 0.2292, Val Loss: 0.2907, Val Accuracy: 87.92%, Val
AUROC: 0.8737, Partial AUROC: 0.1096
Epoch 12/20

Training Epoch 12: 100%|     | 131/131 [01:42<00:00,  1.28it/s,
train_loss=0.203]
Validating Epoch 12: 100%|     | 94/94 [01:17<00:00,  1.21it/s,
val_loss=0.276]

Epoch [11/20], Train Loss: 0.2034, Val Loss: 0.2760, Val Accuracy: 88.86%, Val
AUROC: 0.8748, Partial AUROC: 0.1116
Epoch 13/20

Training Epoch 13: 100%|     | 131/131 [01:55<00:00,  1.13it/s,
train_loss=0.195]
Validating Epoch 13: 100%|     | 94/94 [01:12<00:00,  1.30it/s,
val_loss=0.273]

Epoch [12/20], Train Loss: 0.1950, Val Loss: 0.2727, Val Accuracy: 88.66%, Val
AUROC: 0.8676, Partial AUROC: 0.1076
Epoch 14/20
```

```
Training Epoch 14: 100%|     | 131/131 [01:41<00:00,  1.29it/s,
train_loss=0.166]
Validating Epoch 14: 100%|     | 94/94 [01:12<00:00,  1.30it/s,
val_loss=0.286]

Epoch [13/20], Train Loss: 0.1662, Val Loss: 0.2856, Val Accuracy: 86.58%, Val
AUROC: 0.8741, Partial AUROC: 0.1154
Epoch 15/20

Training Epoch 15: 100%|     | 131/131 [01:47<00:00,  1.22it/s,
train_loss=0.167]
Validating Epoch 15: 100%|     | 94/94 [01:20<00:00,  1.17it/s,
val_loss=0.223]

Epoch [14/20], Train Loss: 0.1669, Val Loss: 0.2230, Val Accuracy: 90.47%, Val
AUROC: 0.8865, Partial AUROC: 0.1277
Epoch 16/20

Training Epoch 16: 100%|     | 131/131 [01:47<00:00,  1.22it/s,
train_loss=0.146]
Validating Epoch 16: 100%|     | 94/94 [01:13<00:00,  1.28it/s,
val_loss=0.219]

Epoch [15/20], Train Loss: 0.1465, Val Loss: 0.2189, Val Accuracy: 90.94%, Val
AUROC: 0.8732, Partial AUROC: 0.1181
Epoch 17/20

Training Epoch 17: 100%|     | 131/131 [01:41<00:00,  1.30it/s,
train_loss=0.151]
Validating Epoch 17: 100%|     | 94/94 [01:21<00:00,  1.15it/s,
val_loss=0.208]

Epoch [16/20], Train Loss: 0.1506, Val Loss: 0.2082, Val Accuracy: 91.81%, Val
AUROC: 0.8843, Partial AUROC: 0.1224
Epoch 18/20

Training Epoch 18: 100%|     | 131/131 [01:45<00:00,  1.24it/s,
train_loss=0.135]
Validating Epoch 18: 100%|     | 94/94 [01:11<00:00,  1.32it/s,
val_loss=0.232]

Epoch [17/20], Train Loss: 0.1350, Val Loss: 0.2320, Val Accuracy: 90.27%, Val
AUROC: 0.8809, Partial AUROC: 0.1202
Epoch 19/20

Training Epoch 19: 100%|     | 131/131 [01:42<00:00,  1.28it/s,
train_loss=0.125]
Validating Epoch 19: 100%|     | 94/94 [01:09<00:00,  1.34it/s,
val_loss=0.217]

Epoch [18/20], Train Loss: 0.1254, Val Loss: 0.2166, Val Accuracy: 90.81%, Val
AUROC: 0.8820, Partial AUROC: 0.1221
Epoch 20/20
```

```
Training Epoch 20: 100%|      | 131/131 [01:48<00:00,  1.20it/s,
train_loss=0.117]
Validating Epoch 20: 100%|      | 94/94 [01:09<00:00,  1.36it/s,
val_loss=0.233]

Epoch [19/20], Train Loss: 0.1170, Val Loss: 0.2328, Val Accuracy: 91.01%, Val
AUROC: 0.8345, Partial AUROC: 0.0921
Best Epoch: 17, Best Validation Loss: 0.2082
Training Complete
```



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

     Class 0       0.98      0.93      0.95      1431
     Class 1       0.22      0.51      0.31        59

    accuracy                           0.91      1490
   macro avg       0.60      0.72      0.63      1490
weighted avg       0.95      0.91      0.93      1490
```

[30]:
```
CustomImageFeatureEfficientNet(
  (efficientnet): Sequential(
    (0): Sequential(
      (0): Conv2dNormActivation(
        (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): SiLU(inplace=True)
      )
      (1): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), groups=32, bias=False)
              (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(32, 8, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(8, 32, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (2): Conv2dNormActivation(
              (0): Conv2d(32, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.0, mode=row)
```

```
        )
      )
      (2): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(16, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(96, 96, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), groups=96, bias=False)
              (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(96, 4, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(4, 96, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(96, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (1): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.0125, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(24, 144, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(144, 144, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=144, bias=False)
              (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(144, 6, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(6, 144, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(144, 24, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.025, mode=row)
      )
    )
    (3): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(24, 144, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(144, 144, kernel_size=(5, 5), stride=(2, 2),
padding=(2, 2), groups=144, bias=False)
            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(144, 6, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(6, 144, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(144, 40, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(40, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.037500000000000006, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(40, 240, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(240, 240, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=240, bias=False)
              (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(240, 10, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(10, 240, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(240, 40, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(40, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.05, mode=row)
        )
      )
      (4): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(40, 240, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
```

```
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(240, 240, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=240, bias=False)
              (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(240, 10, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(10, 240, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(240, 80, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.0625, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(80, 480, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(480, 480, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=480, bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(480, 20, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(20, 480, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
```

```
            (3): Conv2dNormActivation(
              (0): Conv2d(480, 80, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.07500000000000001, mode=row)
        )
        (2): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(80, 480, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(480, 480, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=480, bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(480, 20, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(20, 480, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(480, 80, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.08750000000000001, mode=row)
        )
      )
      (5): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(80, 480, kernel_size=(1, 1), stride=(1, 1),
```

```
bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(480, 480, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=480, bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(480, 20, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(20, 480, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(480, 112, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.1, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(112, 672, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(672, 672, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=672, bias=False)
              (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(672, 28, kernel_size=(1, 1), stride=(1, 1))
```

```
            (fc2): Conv2d(28, 672, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(672, 112, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.1125, mode=row)
      )
      (2): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(112, 672, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(672, 672, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=672, bias=False)
            (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(672, 28, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(28, 672, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(672, 112, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.125, mode=row)
      )
    )
    (6): Sequential(
```

```
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(112, 672, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(672, 672, kernel_size=(5, 5), stride=(2, 2),
padding=(2, 2), groups=672, bias=False)
              (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(672, 28, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(28, 672, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(672, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.1375, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(1152, 1152, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=1152, bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
```

```
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(1152, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.15000000000000002, mode=row)
        )
        (2): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(1152, 1152, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=1152, bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(1152, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
```

```
        (stochastic_depth): StochasticDepth(p=0.1625, mode=row)
      )
      (3): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1152, 1152, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=1152, bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1152, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.17500000000000002, mode=row)
      )
    )
    (7): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1152, 1152, kernel_size=(3, 3), stride=(1, 1),
```

```
padding=(1, 1), groups=1152, bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(1152, 320, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(320, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.1875, mode=row)
        )
      )
      (8): Conv2dNormActivation(
        (0): Conv2d(320, 1280, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(1280, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): SiLU(inplace=True)
      )
    )
    (1): AdaptiveAvgPool2d(output_size=1)
  )
  (fc_image): Linear(in_features=1280, 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)
)
```

5.10 Model 9

[31]:
```
model9 = CustomImageFeatureEfficientNet(feature_input_size=9)  # Assuming 9␣
↪features for metadata
model9.to(device)
# Initialize optimizer
optimizer = optim.Adam(model9.parameters(), lr=0.001)
# Define the loss function with the class weights
criterion = nn.BCELoss()  # Binary classification loss
# Set the number of epochs
```

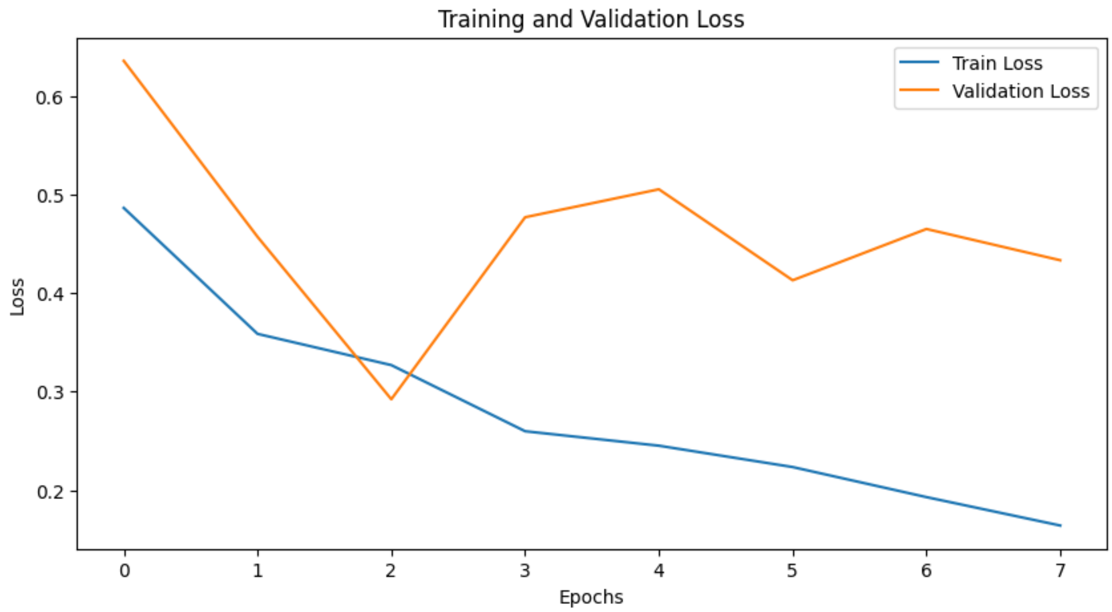
```
epochs = 20
batch_sizes = 16
best_model_path = "best_model9.path"
```

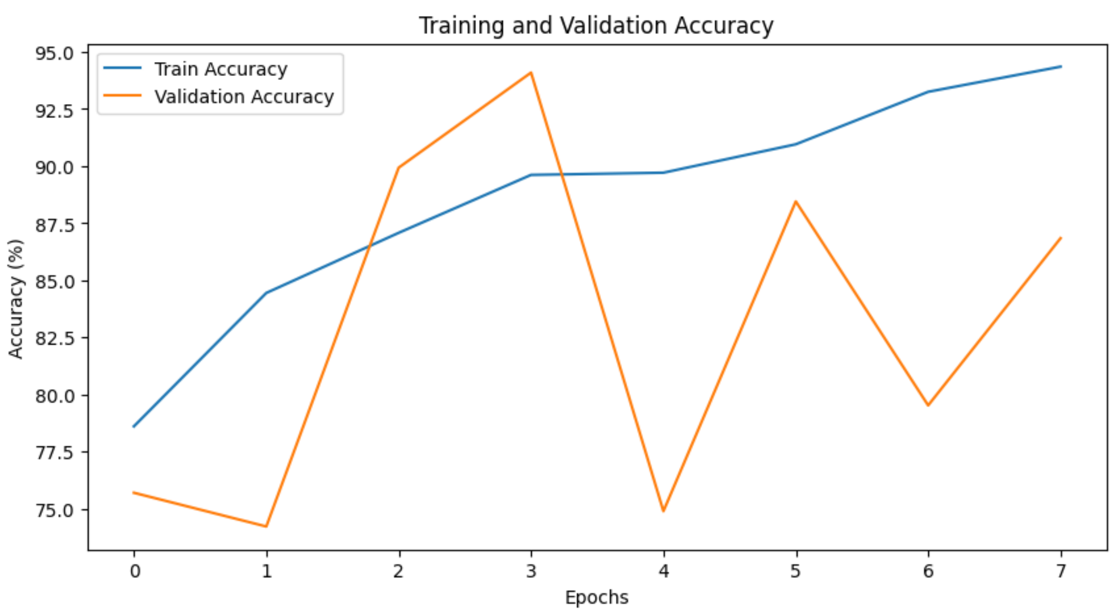
```
/home/jupyter-sohka/.local/lib/python3.10/site-
packages/torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
/home/jupyter-sohka/.local/lib/python3.10/site-
packages/torchvision/models/_utils.py:223: 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=EfficientNet_B0_Weights.IMAGENET1K_V1`. You can also use
`weights=EfficientNet_B0_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

[32]:
```
train_and_validate(model9,effnet_train_dataloader, effnet_val_dataloader,␣
↪criterion, optimizer, epochs, device, best_model_path )
```

```
Epoch 1/20

Training Epoch 1: 100%|      | 131/131 [01:39<00:00,  1.32it/s,
train_loss=0.486]
Validating Epoch 1: 100%|      | 94/94 [01:23<00:00,  1.12it/s,
val_loss=0.636]

Epoch [0/20], Train Loss: 0.4863, Val Loss: 0.6355, Val Accuracy: 75.70%, Val
AUROC: 0.8517, Partial AUROC: 0.1031
Epoch 2/20

Training Epoch 2: 100%|      | 131/131 [01:38<00:00,  1.33it/s,
train_loss=0.359]
Validating Epoch 2: 100%|      | 94/94 [01:14<00:00,  1.27it/s,
val_loss=0.457]

Epoch [1/20], Train Loss: 0.3586, Val Loss: 0.4570, Val Accuracy: 74.23%, Val
AUROC: 0.8024, Partial AUROC: 0.0780
Epoch 3/20

Training Epoch 3: 100%|      | 131/131 [01:42<00:00,  1.28it/s,
train_loss=0.327]
Validating Epoch 3: 100%|      | 94/94 [01:09<00:00,  1.35it/s,
val_loss=0.292]

Epoch [2/20], Train Loss: 0.3269, Val Loss: 0.2922, Val Accuracy: 89.93%, Val
AUROC: 0.8725, Partial AUROC: 0.1104
Epoch 4/20

Training Epoch 4: 100%|      | 131/131 [01:49<00:00,  1.20it/s,
train_loss=0.26]
```

```
Validating Epoch 4: 100%|     | 94/94 [01:08<00:00,  1.37it/s,
val_loss=0.477]

Epoch [3/20], Train Loss: 0.2598, Val Loss: 0.4769, Val Accuracy: 94.09%, Val
AUROC: 0.8231, Partial AUROC: 0.0902
Epoch 5/20

Training Epoch 5: 100%|     | 131/131 [01:39<00:00,  1.31it/s,
train_loss=0.245]
Validating Epoch 5: 100%|     | 94/94 [01:19<00:00,  1.18it/s,
val_loss=0.505]

Epoch [4/20], Train Loss: 0.2451, Val Loss: 0.5052, Val Accuracy: 74.90%, Val
AUROC: 0.8390, Partial AUROC: 0.0930
Epoch 6/20

Training Epoch 6: 100%|     | 131/131 [01:44<00:00,  1.25it/s,
train_loss=0.223]
Validating Epoch 6: 100%|     | 94/94 [01:08<00:00,  1.37it/s,
val_loss=0.413]

Epoch [5/20], Train Loss: 0.2234, Val Loss: 0.4129, Val Accuracy: 88.46%, Val
AUROC: 0.8055, Partial AUROC: 0.0802
Epoch 7/20

Training Epoch 7: 100%|     | 131/131 [01:38<00:00,  1.32it/s,
train_loss=0.193]
Validating Epoch 7: 100%|     | 94/94 [01:18<00:00,  1.19it/s,
val_loss=0.465]

Epoch [6/20], Train Loss: 0.1929, Val Loss: 0.4650, Val Accuracy: 79.53%, Val
AUROC: 0.8621, Partial AUROC: 0.1115
Epoch 8/20

Training Epoch 8: 100%|     | 131/131 [01:39<00:00,  1.32it/s,
train_loss=0.164]
Validating Epoch 8: 100%|     | 94/94 [01:07<00:00,  1.38it/s,
val_loss=0.433]

Epoch [7/20], Train Loss: 0.1641, Val Loss: 0.4332, Val Accuracy: 86.85%, Val
AUROC: 0.8797, Partial AUROC: 0.1251
Early stopping triggered at epoch 7
Best Epoch: 3, Best Validation Loss: 0.2922
Training Complete
```





Classification Report:

	precision	recall	f1-score	support
Class 0	0.99	0.88	0.93	1431
Class 1	0.19	0.69	0.29	59
accuracy			0.87	1490

```
   macro avg       0.59      0.79      0.61      1490
weighted avg       0.95      0.87      0.90      1490
```

[32]:
```
CustomImageFeatureEfficientNet(
  (efficientnet): Sequential(
    (0): Sequential(
      (0): Conv2dNormActivation(
        (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): SiLU(inplace=True)
      )
      (1): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), groups=32, bias=False)
              (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(32, 8, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(8, 32, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (2): Conv2dNormActivation(
              (0): Conv2d(32, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.0, mode=row)
        )
      )
      (2): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(16, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(96, 96, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), groups=96, bias=False)
              (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(96, 4, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(4, 96, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(96, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
              (1): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.0125, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(24, 144, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(144, 144, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=144, bias=False)
              (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(144, 6, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(6, 144, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
```

```
            (3): Conv2dNormActivation(
              (0): Conv2d(144, 24, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.025, mode=row)
        )
      )
      (3): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(24, 144, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(144, 144, kernel_size=(5, 5), stride=(2, 2),
padding=(2, 2), groups=144, bias=False)
              (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(144, 6, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(6, 144, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(144, 40, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(40, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.037500000000000006, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(40, 240, kernel_size=(1, 1), stride=(1, 1),
```

```
bias=False)
              (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(240, 240, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=240, bias=False)
              (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(240, 10, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(10, 240, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(240, 40, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(40, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.05, mode=row)
        )
      )
      (4): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(40, 240, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(240, 240, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=240, bias=False)
              (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
```

```
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(240, 10, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(10, 240, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(240, 80, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.0625, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(80, 480, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(480, 480, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=480, bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(480, 20, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(20, 480, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(480, 80, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.07500000000000001, mode=row)
        )
```

```
        (2): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(80, 480, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(480, 480, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=480, bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(480, 20, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(20, 480, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(480, 80, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.08750000000000001, mode=row)
        )
      )
      (5): Sequential(
        (0): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(80, 480, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(480, 480, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=480, bias=False)
              (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(480, 20, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(20, 480, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(480, 112, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.1, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(112, 672, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(672, 672, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=672, bias=False)
              (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(672, 28, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(28, 672, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(672, 112, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
          )
        )
        (stochastic_depth): StochasticDepth(p=0.1125, mode=row)
      )
      (2): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(112, 672, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(672, 672, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=672, bias=False)
            (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(672, 28, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(28, 672, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(672, 112, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.125, mode=row)
      )
    )
    (6): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(112, 672, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
```

```
            (1): Conv2dNormActivation(
              (0): Conv2d(672, 672, kernel_size=(5, 5), stride=(2, 2),
padding=(2, 2), groups=672, bias=False)
              (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(672, 28, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(28, 672, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(672, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.1375, mode=row)
        )
        (1): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(1152, 1152, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=1152, bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
```

```
              (0): Conv2d(1152, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.15000000000000002, mode=row)
        )
        (2): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (1): Conv2dNormActivation(
              (0): Conv2d(1152, 1152, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=1152, bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
              (2): SiLU(inplace=True)
            )
            (2): SqueezeExcitation(
              (avgpool): AdaptiveAvgPool2d(output_size=1)
              (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
              (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(1152, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.1625, mode=row)
        )
        (3): MBConv(
          (block): Sequential(
            (0): Conv2dNormActivation(
              (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1152, 1152, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=1152, bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1152, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.17500000000000002, mode=row)
      )
    )
    (7): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1152, 1152, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1152, bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
```

```
              (activation): SiLU(inplace=True)
              (scale_activation): Sigmoid()
            )
            (3): Conv2dNormActivation(
              (0): Conv2d(1152, 320, kernel_size=(1, 1), stride=(1, 1),
bias=False)
              (1): BatchNorm2d(320, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
          )
          (stochastic_depth): StochasticDepth(p=0.1875, mode=row)
        )
      )
      (8): Conv2dNormActivation(
        (0): Conv2d(320, 1280, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(1280, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (2): SiLU(inplace=True)
      )
    )
    (1): AdaptiveAvgPool2d(output_size=1)
  )
  (fc_image): Linear(in_features=1280, 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)
)
```

6 Select Winning Model

Based on the performance metrics of the 9 models, Model 7 has been selected as the winning model. This decision was made after evaluating each model's performance on key metrics such as accuracy, AUROC, partial AUC, loss, precision, and recall.

The next step is to evaluate Model 7 on the test data, which contains unseen data that was not used during training or validation. This step is essential to assess the model's ability to generalize to new, real-world cases.

[18]:
```
effnet_test_dataset = MultiInputDataset(hdf5_file='../data/raw/test_image.
 ↪hdf5', csv_file='../data/processed/processed-test-metadata1.csv',␣
 ↪transform=get_normal_transform(resize_size=(224,224)))
# Create a DataLoader
effnet_test_dataloader = DataLoader(effnet_test_dataset,  batch_size=64,␣
 ↪shuffle=True)
```

[19]:
```
final_model = CustomImageFeatureEfficientNet(9)
final_model_path = "best_model7.pth"
```

```python
final_model.load_state_dict(torch.load(final_model_path, map_location=torch.
 ↪device('cpu')))

final_model.eval()
all_labels, all_probs = [], []
with torch.no_grad():
    for images, metadata, labels in effnet_test_dataloader:
        images, metadata, labels = images.to(device), metadata.to(device),␣
 ↪labels.float().to(device).unsqueeze(1)
        probs = final_model(images,metadata)
        all_labels.extend(labels.cpu().numpy())
        all_probs.extend(probs.cpu().numpy())
        predicted = (probs > 0.5).float()

    partial_auroc=score(np.array(all_labels),np.array(all_probs))
    print(f'The partial auroc of the final model on the test image is␣
 ↪{partial_auroc}')
    print(classification_report(all_labels, (np.array(all_probs) >= 0.5).
 ↪astype(int), target_names=['Class 0', 'Class 1']))
```

```
/home/jupyter-sohka/.local/lib/python3.10/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(
/home/jupyter-sohka/.local/lib/python3.10/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=EfficientNet_B0_Weights.IMAGENET1K_V1`. You can also use
`weights=EfficientNet_B0_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
/tmp/ipykernel_3535228/1883220575.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.
  final_model.load_state_dict(torch.load(final_model_path,
map_location=torch.device('cpu')))

The partial auroc of the final model on the test image is 0.11427590046074211
              precision    recall  f1-score   support

     Class 0       0.98      0.93      0.96      1431
     Class 1       0.26      0.63      0.37        59

    accuracy                           0.92      1490
   macro avg       0.62      0.78      0.66      1490
weighted avg       0.96      0.92      0.93      1490
```

As I expected, comparing the performance metrics on the test data versus the validation data for Model 7 reveals an improvement in the recall and F1-score for Class 1. This is a significant observation because it indicates that the model generalizes well to unseen data, which is crucial for real-world applications. Additionally, the partial AUC-above-TPR also shows improvement compared to the best epoch of Model 7 during validation. This suggests that the model performs better in capturing true positive malignant cases in regions of high true positive rates (TPR), which aligns with the primary goal of detecting malignant skin lesions effectively. These results demonstrate that the model is not overfitting to the validation set and is capable of making accurate predictions on new, unseen data.

[ ]:

```
```