HW5_BreastMNIST_Classification

January 17, 2024

1 HW5 - Binary Classification using Swin Transformer

Intelligent Analysis of Biomedical Images

Fall 2023

- Name: Javad Razi
- Student id: 401204354

Introduction:

In this educational notebook, we will explore binary classification on the BreastMNIST dataset, consisting of 780 breast ultrasound images. This task involves using the Swin Transformer, a cutting-edge neural network model, to distinguish between benign (including normal) and malignant cases.

We'll tackle the common challenge of class imbalance in medical datasets and learn to improve model performance using class weights. Additionally, we'll delve into evaluating our model with ROC curves and AUC, essential tools for assessing performance in medical image classification.

1.1 Packages & Modules

[ ]:
```
from IPython.display import clear_output

import torch
from tqdm import tqdm
import torch.nn as nn
from torch.utils.data import DataLoader
from torchvision import transforms
import matplotlib.pyplot as plt
from collections import Counter


!pip install torchmetrics
!pip install timm
!pip install medmnist
```

```
Collecting torchmetrics
  Downloading torchmetrics-1.3.0-py3-none-any.whl (840 kB)
```

```
                                        840.2/840.2
kB 5.2 MB/s eta 0:00:00
Requirement already satisfied: numpy>1.20.0 in
/usr/local/lib/python3.10/dist-packages (from torchmetrics) (1.23.5)
Requirement already satisfied: packaging>17.1 in /usr/local/lib/python3.10/dist-
packages (from torchmetrics) (23.2)
Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/python3.10/dist-
packages (from torchmetrics) (2.1.0+cu121)
Collecting lightning-utilities>=0.8.0 (from torchmetrics)
  Downloading lightning_utilities-0.10.0-py3-none-any.whl (24 kB)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from lightning-utilities>=0.8.0->torchmetrics) (67.7.2)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.10/dist-packages (from lightning-
utilities>=0.8.0->torchmetrics) (4.5.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from torch>=1.10.0->torchmetrics) (3.13.1)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
(from torch>=1.10.0->torchmetrics) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-
packages (from torch>=1.10.0->torchmetrics) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch>=1.10.0->torchmetrics) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch>=1.10.0->torchmetrics) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-
packages (from torch>=1.10.0->torchmetrics) (2.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from
jinja2->torch>=1.10.0->torchmetrics) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-
packages (from sympy->torch>=1.10.0->torchmetrics) (1.3.0)
Installing collected packages: lightning-utilities, torchmetrics
Successfully installed lightning-utilities-0.10.0 torchmetrics-1.3.0
Collecting timm
  Downloading timm-0.9.12-py3-none-any.whl (2.2 MB)
                                        2.2/2.2 MB
9.7 MB/s eta 0:00:00
Requirement already satisfied: torch>=1.7 in
/usr/local/lib/python3.10/dist-packages (from timm) (2.1.0+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-
packages (from timm) (0.16.0+cu121)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages
(from timm) (6.0.1)
Requirement already satisfied: huggingface-hub in
/usr/local/lib/python3.10/dist-packages (from timm) (0.20.2)
Requirement already satisfied: safetensors in /usr/local/lib/python3.10/dist-
```

```
packages (from timm) (0.4.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from torch>=1.7->timm) (3.13.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.10/dist-packages (from torch>=1.7->timm) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
(from torch>=1.7->timm) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-
packages (from torch>=1.7->timm) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch>=1.7->timm) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch>=1.7->timm) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-
packages (from torch>=1.7->timm) (2.1.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from huggingface-hub->timm) (2.31.0)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-
packages (from huggingface-hub->timm) (4.66.1)
Requirement already satisfied: packaging>=20.9 in
/usr/local/lib/python3.10/dist-packages (from huggingface-hub->timm) (23.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from torchvision->timm) (1.23.5)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in
/usr/local/lib/python3.10/dist-packages (from torchvision->timm) (9.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.7->timm) (2.1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-hub->timm)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->huggingface-hub->timm) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-hub->timm)
(2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->huggingface-hub->timm)
(2023.11.17)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-
packages (from sympy->torch>=1.7->timm) (1.3.0)
Installing collected packages: timm
Successfully installed timm-0.9.12
Collecting medmnist
  Downloading medmnist-2.2.4-py3-none-any.whl (22 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from medmnist) (1.23.5)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from medmnist) (1.5.3)
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
packages (from medmnist) (1.2.2)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.10/dist-
packages (from medmnist) (0.19.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from medmnist) (4.66.1)
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages
(from medmnist) (9.4.0)
Collecting fire (from medmnist)
  Downloading fire-0.5.0.tar.gz (88 kB)
                            88.3/88.3 kB
3.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages
(from medmnist) (2.1.0+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-
packages (from medmnist) (0.16.0+cu121)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from fire->medmnist) (1.16.0)
Requirement already satisfied: termcolor in /usr/local/lib/python3.10/dist-
packages (from fire->medmnist) (2.4.0)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->medmnist) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->medmnist) (2023.3.post1)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-image->medmnist) (1.11.4)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-
packages (from scikit-image->medmnist) (3.2.1)
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-image->medmnist) (2.31.6)
Requirement already satisfied: tifffile>=2019.7.26 in
/usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist)
(2023.12.9)
Requirement already satisfied: PyWavelets>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist) (1.5.0)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist) (23.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn->medmnist) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn->medmnist) (3.2.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from torch->medmnist) (3.13.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.10/dist-packages (from torch->medmnist) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
(from torch->medmnist) (1.12)
```

```
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch->medmnist) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch->medmnist) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-
packages (from torch->medmnist) (2.1.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from torchvision->medmnist) (2.31.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch->medmnist) (2.1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->torchvision->medmnist)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->torchvision->medmnist) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->torchvision->medmnist)
(2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->torchvision->medmnist)
(2023.11.17)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-
packages (from sympy->torch->medmnist) (1.3.0)
Building wheels for collected packages: fire
  Building wheel for fire (setup.py) … done
  Created wheel for fire: filename=fire-0.5.0-py2.py3-none-any.whl size=116934
sha256=39652454730d5fa549d14c0d228b4536affcaa9bde359a20c17b41d8e5d6818e
  Stored in directory: /root/.cache/pip/wheels/90/d4/f7/9404e5db0116bd4d43e5666e
aa3e70ab53723e1e3ea40c9a95
Successfully built fire
Installing collected packages: fire, medmnist
Successfully installed fire-0.5.0 medmnist-2.2.4
```

[ ]:
```python
from torchmetrics import Accuracy
from timm import create_model
import medmnist

device = "cuda" if torch.cuda.is_available() else "cpu"
clear_output()
```

1.2 Data

The BreastMNIST dataset is simplified into a binary classification problem, merging normal and benign images into a single class, contrasting against malignant images. The dataset, originally $1\times500\times500$ in size, is resized to $1\times28\times28$ and split into training, validation, and test sets. We address the class imbalance by applying class weights in our loss function, enhancing the focus on underrepresented classes.

[ ]:
```
info = medmnist.INFO['breastmnist']
n_channels = info['n_channels']

DataClass = getattr(medmnist, info['python_class'])
```

[ ]:
```
print(info['n_samples'])
print(info['label'])
```

```
{'train': 546, 'val': 78, 'test': 156}
{'0': 'malignant', '1': 'normal, benign'}
```

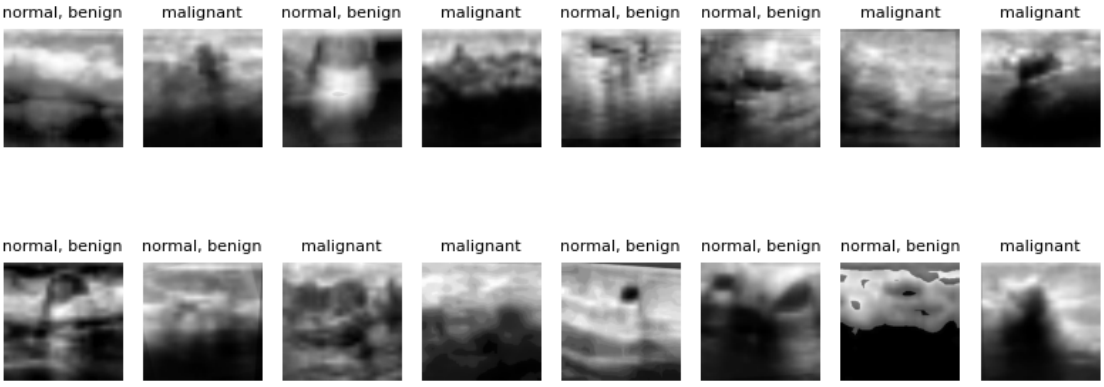
[ ]:
```
# preprocessing
train_transform = transforms.Compose([
    transforms.Resize(224),
    transforms.AugMix(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[.5], std=[.5])
])
test_transform = transforms.Compose([
    transforms.Resize(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[.5], std=[.5])
])



BATCH_SIZE = 16

# load the data
train_dataset = DataClass(split='train', transform=train_transform,␣
 ↪download=True)
val_dataset = DataClass(split='val', transform=train_transform, download=True)
test_dataset = DataClass(split='test', transform=test_transform, download=True)

# Create instances of your dataset and data loader
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)
```

```
Downloading https://zenodo.org/records/10519195/files/breastmnist.npz?download=1
to /root/.medmnist/breastmnist.npz
```

```
100%|      | 559580/559580 [00:01<00:00, 433960.85it/s]
```

```
Using downloaded and verified file: /root/.medmnist/breastmnist.npz
Using downloaded and verified file: /root/.medmnist/breastmnist.npz
```

[ ]:
```
images, labels = next(iter(train_loader))
images = images[:16]
labels = labels[:16]

fig, axes = plt.subplots(2, 8, figsize=(8, 4))

for i, ax in enumerate(axes.flatten()):
    # Convert the tensor image to numpy array format
    image = images[i].numpy().transpose(1, 2, 0)
    # Normalize the image to the range [0, 1]
    image = (image + 1) / 2
    ax.imshow(image, cmap='gray')
    ax.axis("off")
    ax.set_title(f"{info['label'][str(labels[i].item())]}", fontsize=8)

plt.tight_layout()
plt.show()
```



1.2.1 Train

[ ]:
```
model = create_model(model_name="swin_tiny_patch4_window7_224",
                         pretrained=True,
                         num_classes=1,
                         in_chans=n_channels)
model = model.to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

```
/usr/local/lib/python3.10/dist-packages/torch/functional.py:504: UserWarning:
torch.meshgrid: in an upcoming release, it will be required to pass the indexing
argument. (Triggered internally at
../aten/src/ATen/native/TensorShape.cpp:3526.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88:
```

```
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(

model.safetensors:   0%|          | 0.00/114M [00:00<?, ?B/s]
```

[ ]:
```python
from torchmetrics.classification import BinaryAccuracy, BinaryF1Score,␣
↪BinaryConfusionMatrix

def train_one_epoch(model, train_loader, loss_fn, optimizer, device,␣
↪epoch=None):
    model.train()
    total_loss = 0.0
    total_samples = 0
    accuracy_metric = BinaryAccuracy().to(device)
    f1_score_metric = BinaryF1Score().to(device)
    confusion_matrix_metric = BinaryConfusionMatrix().to(device)

    for batch in train_loader:
        inputs, labels = batch
        inputs, labels = inputs.to(device, dtype=torch.float), labels.
↪to(device, dtype=torch.float)

        optimizer.zero_grad()
        outputs = model(inputs)

        # Squeeze labels to make sure they are in the correct shape for␣
↪BCEWithLogitsLoss
        labels = labels.squeeze(1)

        # Compute loss
        loss = loss_fn(outputs, labels.unsqueeze(1))  # Unsqueeze labels for␣
↪BCEWithLogitsLoss

        loss.backward()
        optimizer.step()

        total_loss += loss.item() * inputs.size(0)
        total_samples += inputs.size(0)

        # Update metrics
```

```
        preds = torch.sigmoid(outputs) > 0.5
        preds = preds.squeeze().int()  # Squeeze and convert to int for metrics

        # Ensure labels are squeezed to match preds shape for metric updates
        labels = labels.int()  # Convert to int for metrics

        accuracy_metric.update(preds, labels)
        f1_score_metric.update(preds, labels)
        confusion_matrix_metric.update(preds, labels)

    avg_loss = total_loss / total_samples
    avg_acc = accuracy_metric.compute()
    avg_f1 = f1_score_metric.compute()
    cm = confusion_matrix_metric.compute()

    return model, avg_loss, avg_acc, avg_f1, cm
```

[ ]:
```
def validate_one_epoch(model, val_loader, loss_fn, device, epoch=None):
    model.eval()
    total_loss = 0.0
    n_samples = 0
    accuracy_metric = BinaryAccuracy().to(device)
    f1_score_metric = BinaryF1Score().to(device)
    confusion_matrix_metric = BinaryConfusionMatrix().to(device)

    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device).float()

            outputs = model(inputs)

            # Ensure labels are squeezed to match output shape for loss␣
 ↪calculation
            labels = labels.squeeze(1)

            # Calculate loss
            loss = loss_fn(outputs, labels.unsqueeze(1))  # Unsqueeze labels␣
 ↪for BCEWithLogitsLoss

            total_loss += loss.item() * inputs.size(0)
            n_samples += inputs.size(0)

            # Update metrics
            preds = torch.sigmoid(outputs) > 0.5
            preds = preds.squeeze().int()  # Convert to int for metrics

            # Ensure labels are squeezed to match preds shape for metric updates
```

```
            labels = labels.int()  # Convert to int for metrics

            accuracy_metric.update(preds, labels)
            f1_score_metric.update(preds, labels)
            confusion_matrix_metric.update(preds, labels)

    avg_loss = total_loss / n_samples
    avg_acc = accuracy_metric.compute()
    avg_f1 = f1_score_metric.compute()
    cm = confusion_matrix_metric.compute()

    return avg_loss, avg_acc, avg_f1, cm
```

[ ]:
```
import os
import torch
import pickle

# Save the best model and history in a training session
def save_if_best_model(history, model, model_name, metric='loss', model_dir='./
 ↪models'):
    metric_values = history[f'{metric}_valid']
    best_metric_value = min(metric_values) if metric == 'loss' else␣
 ↪max(metric_values)

    # Get the paths for the model and history
    os.makedirs(model_dir, exist_ok=True)
    model_path = os.path.join(model_dir, f'{model_name}.pt')
    history_path = os.path.join(model_dir, f'{model_name}_history.pkl')

    # Save the model and history if it's the best one based on the metric
    if history['best_metric'] is None or \
       (metric == 'loss' and best_metric_value < history['best_metric']) or \
       (metric != 'loss' and best_metric_value > history['best_metric']):
        history['best_metric'] = best_metric_value
        torch.save(model.state_dict(), model_path)
        with open(history_path, 'wb') as f:
            pickle.dump(history, f)
        print(f"Best model saved. {metric}: {best_metric_value}")

# Load the model and history
def load_model(model, model_name, model_dir='./models'):
    model_path = os.path.join(model_dir, f'{model_name}.pt')
    history_path = os.path.join(model_dir, f'{model_name}_history.pkl')
    loaded_model = None
    history = None

    if os.path.exists(model_path):
```

```
        model.load_state_dict(torch.load(model_path))
        print(f"Model loaded from {model_path}")
        loaded_model = model
    else:
        print(f"No saved model found at {model_path}")

    if os.path.exists(history_path):
        with open(history_path, 'rb') as f:
            history = pickle.load(f)
        print(f"History loaded from {history_path}")

    return loaded_model, history
```

[ ]:
```
from collections import defaultdict

def train_model(model, model_name, loss_fn, optimizer, device, metric='loss',␣
 ↪n_epochs=100):
    history = defaultdict(list)
    history['best_metric'] = None

    # Training Loop
    for epoch in range(n_epochs):
        torch.cuda.empty_cache()

        # Train one epoch
        model, loss_train, acc_train, f1_train, cm_train = train_one_epoch(
            model, train_loader, loss_fn, optimizer, device, epoch
        )

        # Validate one epoch
        loss_valid, acc_valid, f1_valid, cm_valid = validate_one_epoch(
            model, val_loader, loss_fn, device, epoch
        )

        # Append metrics to history
        history['loss_train'].append(loss_train)
        history['loss_valid'].append(loss_valid)
        history['acc_train'].append(acc_train)
        history['acc_valid'].append(acc_valid)
        history['f1_train'].append(f1_train)
        history['f1_valid'].append(f1_valid)
        history['cm_train'].append(cm_train)
        history['cm_valid'].append(cm_valid)

        # Save the model if it has the best performance yet, based on the␣
 ↪provided metric
        save_if_best_model(history, model, model_name, metric)
```

```
        print(f"Epoch: {epoch+1}, Train Loss: {loss_train:.4f}, Train Accuracy:␣
 ↪{acc_train:.4f}, "
              f"Train F1: {f1_train:.4f}, Valid Loss: {loss_valid:.4f}, Valid␣
 ↪Accuracy: {acc_valid:.4f}, "
              f"Valid F1: {f1_valid:.4f}")

    return history

bce_loss = nn.BCEWithLogitsLoss()

# Used validation accuracy as the metric to save the best model, as the current␣
 ↪section seems to emphasize on the shortcommings of accuracy as a metric
vanilla_train_history = train_model(model=model,␣
 ↪model_name='vanilla_bcm_classifier', loss_fn=bce_loss, optimizer=optimizer,␣
 ↪device=device, metric='acc', n_epochs=100)
```

```
Best model saved. acc: 0.7435897588729858
Epoch: 1, Train Loss: 0.5578, Train Accuracy: 0.7326, Train F1: 0.8423, Valid
Loss: 0.4953, Valid Accuracy: 0.7436, Valid F1: 0.8507
Best model saved. acc: 0.8205128312110901
Epoch: 2, Train Loss: 0.5359, Train Accuracy: 0.7527, Train F1: 0.8508, Valid
Loss: 0.4902, Valid Accuracy: 0.8205, Valid F1: 0.8852
Best model saved. acc: 0.8461538553237915
Epoch: 3, Train Loss: 0.4860, Train Accuracy: 0.7930, Train F1: 0.8720, Valid
Loss: 0.3792, Valid Accuracy: 0.8462, Valid F1: 0.9032
Best model saved. acc: 0.8717948794364929
Epoch: 4, Train Loss: 0.4448, Train Accuracy: 0.7985, Train F1: 0.8706, Valid
Loss: 0.3872, Valid Accuracy: 0.8718, Valid F1: 0.9180
Epoch: 5, Train Loss: 0.4457, Train Accuracy: 0.8004, Train F1: 0.8695, Valid
Loss: 0.4756, Valid Accuracy: 0.7821, Valid F1: 0.8411
Best model saved. acc: 0.9102563858032227
Epoch: 6, Train Loss: 0.4596, Train Accuracy: 0.7930, Train F1: 0.8709, Valid
Loss: 0.3517, Valid Accuracy: 0.9103, Valid F1: 0.9412
Epoch: 7, Train Loss: 0.3775, Train Accuracy: 0.8608, Train F1: 0.9089, Valid
Loss: 0.4248, Valid Accuracy: 0.8333, Valid F1: 0.8807
Epoch: 8, Train Loss: 0.3523, Train Accuracy: 0.8681, Train F1: 0.9126, Valid
Loss: 0.3294, Valid Accuracy: 0.8718, Valid F1: 0.9153
Epoch: 9, Train Loss: 0.3986, Train Accuracy: 0.8370, Train F1: 0.8929, Valid
Loss: 0.3506, Valid Accuracy: 0.8846, Valid F1: 0.9268
Epoch: 10, Train Loss: 0.3518, Train Accuracy: 0.8516, Train F1: 0.9032, Valid
Loss: 0.2754, Valid Accuracy: 0.8846, Valid F1: 0.9244
Epoch: 11, Train Loss: 0.3241, Train Accuracy: 0.8626, Train F1: 0.9091, Valid
Loss: 0.2742, Valid Accuracy: 0.9103, Valid F1: 0.9421
Epoch: 12, Train Loss: 0.3101, Train Accuracy: 0.8700, Train F1: 0.9139, Valid
Loss: 0.3018, Valid Accuracy: 0.8846, Valid F1: 0.9268
Epoch: 13, Train Loss: 0.2916, Train Accuracy: 0.8883, Train F1: 0.9262, Valid
```

```
Loss: 0.2929, Valid Accuracy: 0.8974, Valid F1: 0.9298
Epoch: 14, Train Loss: 0.2618, Train Accuracy: 0.8938, Train F1: 0.9296, Valid
Loss: 0.3294, Valid Accuracy: 0.8590, Valid F1: 0.9009
Epoch: 15, Train Loss: 0.2850, Train Accuracy: 0.8846, Train F1: 0.9223, Valid
Loss: 0.3143, Valid Accuracy: 0.8718, Valid F1: 0.9107
Best model saved. acc: 0.9230769276618958
Epoch: 16, Train Loss: 0.2443, Train Accuracy: 0.9176, Train F1: 0.9451, Valid
Loss: 0.2644, Valid Accuracy: 0.9231, Valid F1: 0.9483
Epoch: 17, Train Loss: 0.2006, Train Accuracy: 0.9194, Train F1: 0.9451, Valid
Loss: 0.3236, Valid Accuracy: 0.9103, Valid F1: 0.9421
Epoch: 18, Train Loss: 0.4485, Train Accuracy: 0.8297, Train F1: 0.8902, Valid
Loss: 0.2839, Valid Accuracy: 0.9103, Valid F1: 0.9402
Epoch: 19, Train Loss: 0.2314, Train Accuracy: 0.9084, Train F1: 0.9392, Valid
Loss: 0.2738, Valid Accuracy: 0.8974, Valid F1: 0.9322
Epoch: 20, Train Loss: 0.2361, Train Accuracy: 0.9084, Train F1: 0.9390, Valid
Loss: 0.2655, Valid Accuracy: 0.8974, Valid F1: 0.9310
Epoch: 21, Train Loss: 0.2045, Train Accuracy: 0.9286, Train F1: 0.9524, Valid
Loss: 0.3097, Valid Accuracy: 0.8846, Valid F1: 0.9256
Epoch: 22, Train Loss: 0.1819, Train Accuracy: 0.9341, Train F1: 0.9557, Valid
Loss: 0.2607, Valid Accuracy: 0.9103, Valid F1: 0.9412
Epoch: 23, Train Loss: 0.1373, Train Accuracy: 0.9579, Train F1: 0.9716, Valid
Loss: 0.4263, Valid Accuracy: 0.8974, Valid F1: 0.9322
Epoch: 24, Train Loss: 0.1757, Train Accuracy: 0.9322, Train F1: 0.9542, Valid
Loss: 0.2994, Valid Accuracy: 0.8974, Valid F1: 0.9322
Epoch: 25, Train Loss: 0.1410, Train Accuracy: 0.9505, Train F1: 0.9666, Valid
Loss: 0.2960, Valid Accuracy: 0.9103, Valid F1: 0.9381
Epoch: 26, Train Loss: 0.1727, Train Accuracy: 0.9396, Train F1: 0.9592, Valid
Loss: 0.3489, Valid Accuracy: 0.8718, Valid F1: 0.9138
Epoch: 27, Train Loss: 0.1419, Train Accuracy: 0.9542, Train F1: 0.9691, Valid
Loss: 0.3567, Valid Accuracy: 0.8846, Valid F1: 0.9231
Epoch: 28, Train Loss: 0.1057, Train Accuracy: 0.9615, Train F1: 0.9740, Valid
Loss: 0.2969, Valid Accuracy: 0.9231, Valid F1: 0.9483
Epoch: 29, Train Loss: 0.1067, Train Accuracy: 0.9579, Train F1: 0.9714, Valid
Loss: 0.4971, Valid Accuracy: 0.8333, Valid F1: 0.8870
Epoch: 30, Train Loss: 0.1045, Train Accuracy: 0.9560, Train F1: 0.9701, Valid
Loss: 0.3748, Valid Accuracy: 0.8718, Valid F1: 0.9138
Epoch: 31, Train Loss: 0.1021, Train Accuracy: 0.9634, Train F1: 0.9750, Valid
Loss: 0.4641, Valid Accuracy: 0.8974, Valid F1: 0.9333
Epoch: 32, Train Loss: 0.1806, Train Accuracy: 0.9304, Train F1: 0.9532, Valid
Loss: 0.2644, Valid Accuracy: 0.9231, Valid F1: 0.9483
Best model saved. acc: 0.9487179517745972
Epoch: 33, Train Loss: 0.1146, Train Accuracy: 0.9432, Train F1: 0.9616, Valid
Loss: 0.2691, Valid Accuracy: 0.9487, Valid F1: 0.9655
Epoch: 34, Train Loss: 0.0736, Train Accuracy: 0.9744, Train F1: 0.9827, Valid
Loss: 0.2749, Valid Accuracy: 0.9359, Valid F1: 0.9573
Epoch: 35, Train Loss: 0.0926, Train Accuracy: 0.9744, Train F1: 0.9825, Valid
Loss: 0.3870, Valid Accuracy: 0.9103, Valid F1: 0.9402
Epoch: 36, Train Loss: 0.0888, Train Accuracy: 0.9744, Train F1: 0.9825, Valid
```

```
Loss: 0.3501, Valid Accuracy: 0.9103, Valid F1: 0.9391
Epoch: 37, Train Loss: 0.1036, Train Accuracy: 0.9579, Train F1: 0.9716, Valid
Loss: 0.4002, Valid Accuracy: 0.8590, Valid F1: 0.9009
Epoch: 38, Train Loss: 0.1158, Train Accuracy: 0.9542, Train F1: 0.9689, Valid
Loss: 0.3282, Valid Accuracy: 0.8974, Valid F1: 0.9310
Epoch: 39, Train Loss: 0.1027, Train Accuracy: 0.9560, Train F1: 0.9699, Valid
Loss: 0.5654, Valid Accuracy: 0.8846, Valid F1: 0.9256
Epoch: 40, Train Loss: 0.1586, Train Accuracy: 0.9396, Train F1: 0.9594, Valid
Loss: 0.2482, Valid Accuracy: 0.9103, Valid F1: 0.9391
Epoch: 41, Train Loss: 0.0746, Train Accuracy: 0.9799, Train F1: 0.9863, Valid
Loss: 0.3069, Valid Accuracy: 0.9103, Valid F1: 0.9391
Epoch: 42, Train Loss: 0.1016, Train Accuracy: 0.9652, Train F1: 0.9763, Valid
Loss: 0.3470, Valid Accuracy: 0.8974, Valid F1: 0.9310
Epoch: 43, Train Loss: 0.0726, Train Accuracy: 0.9744, Train F1: 0.9825, Valid
Loss: 0.3688, Valid Accuracy: 0.8974, Valid F1: 0.9310
Epoch: 44, Train Loss: 0.0497, Train Accuracy: 0.9780, Train F1: 0.9850, Valid
Loss: 0.3102, Valid Accuracy: 0.9359, Valid F1: 0.9573
Epoch: 45, Train Loss: 0.0547, Train Accuracy: 0.9762, Train F1: 0.9839, Valid
Loss: 0.2391, Valid Accuracy: 0.9359, Valid F1: 0.9558
Epoch: 46, Train Loss: 0.0574, Train Accuracy: 0.9817, Train F1: 0.9875, Valid
Loss: 0.2876, Valid Accuracy: 0.9231, Valid F1: 0.9483
Epoch: 47, Train Loss: 0.0418, Train Accuracy: 0.9890, Train F1: 0.9925, Valid
Loss: 0.4617, Valid Accuracy: 0.8974, Valid F1: 0.9298
Epoch: 48, Train Loss: 0.0341, Train Accuracy: 0.9835, Train F1: 0.9888, Valid
Loss: 0.3774, Valid Accuracy: 0.9103, Valid F1: 0.9391
Epoch: 49, Train Loss: 0.0327, Train Accuracy: 0.9853, Train F1: 0.9900, Valid
Loss: 0.4299, Valid Accuracy: 0.8974, Valid F1: 0.9344
Epoch: 50, Train Loss: 0.0507, Train Accuracy: 0.9835, Train F1: 0.9887, Valid
Loss: 0.3435, Valid Accuracy: 0.9231, Valid F1: 0.9500
Epoch: 51, Train Loss: 0.0338, Train Accuracy: 0.9890, Train F1: 0.9925, Valid
Loss: 0.4395, Valid Accuracy: 0.9103, Valid F1: 0.9402
Epoch: 52, Train Loss: 0.0329, Train Accuracy: 0.9872, Train F1: 0.9913, Valid
Loss: 0.4123, Valid Accuracy: 0.9103, Valid F1: 0.9402
Epoch: 53, Train Loss: 0.0900, Train Accuracy: 0.9634, Train F1: 0.9751, Valid
Loss: 0.4471, Valid Accuracy: 0.9359, Valid F1: 0.9573
Epoch: 54, Train Loss: 0.0642, Train Accuracy: 0.9762, Train F1: 0.9838, Valid
Loss: 0.5897, Valid Accuracy: 0.8974, Valid F1: 0.9298
Epoch: 55, Train Loss: 0.0748, Train Accuracy: 0.9707, Train F1: 0.9799, Valid
Loss: 0.4591, Valid Accuracy: 0.9103, Valid F1: 0.9402
Epoch: 56, Train Loss: 0.0254, Train Accuracy: 0.9927, Train F1: 0.9950, Valid
Loss: 0.4954, Valid Accuracy: 0.8846, Valid F1: 0.9204
Epoch: 57, Train Loss: 0.0558, Train Accuracy: 0.9872, Train F1: 0.9913, Valid
Loss: 0.5527, Valid Accuracy: 0.8718, Valid F1: 0.9138
Epoch: 58, Train Loss: 0.0413, Train Accuracy: 0.9853, Train F1: 0.9900, Valid
Loss: 0.3324, Valid Accuracy: 0.9103, Valid F1: 0.9412
Epoch: 59, Train Loss: 0.0330, Train Accuracy: 0.9890, Train F1: 0.9925, Valid
Loss: 0.3834, Valid Accuracy: 0.8974, Valid F1: 0.9310
Epoch: 60, Train Loss: 0.0305, Train Accuracy: 0.9890, Train F1: 0.9925, Valid
```

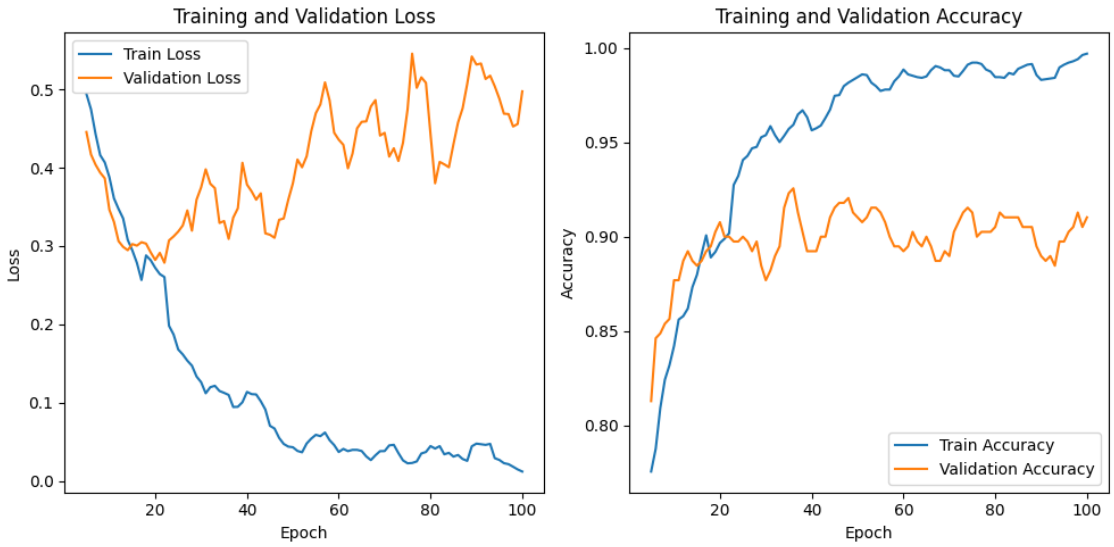
```
Loss: 0.4145, Valid Accuracy: 0.8974, Valid F1: 0.9322
Epoch: 61, Train Loss: 0.0453, Train Accuracy: 0.9799, Train F1: 0.9863, Valid
Loss: 0.4628, Valid Accuracy: 0.8974, Valid F1: 0.9322
Epoch: 62, Train Loss: 0.0416, Train Accuracy: 0.9835, Train F1: 0.9888, Valid
Loss: 0.4027, Valid Accuracy: 0.9103, Valid F1: 0.9391
Epoch: 63, Train Loss: 0.0494, Train Accuracy: 0.9817, Train F1: 0.9875, Valid
Loss: 0.4266, Valid Accuracy: 0.8846, Valid F1: 0.9231
Epoch: 64, Train Loss: 0.0329, Train Accuracy: 0.9872, Train F1: 0.9912, Valid
Loss: 0.5445, Valid Accuracy: 0.8846, Valid F1: 0.9256
Epoch: 65, Train Loss: 0.0231, Train Accuracy: 0.9927, Train F1: 0.9950, Valid
Loss: 0.4559, Valid Accuracy: 0.9231, Valid F1: 0.9483
Epoch: 66, Train Loss: 0.0107, Train Accuracy: 0.9963, Train F1: 0.9975, Valid
Loss: 0.4656, Valid Accuracy: 0.8718, Valid F1: 0.9138
Epoch: 67, Train Loss: 0.0182, Train Accuracy: 0.9945, Train F1: 0.9962, Valid
Loss: 0.4978, Valid Accuracy: 0.8718, Valid F1: 0.9123
Epoch: 68, Train Loss: 0.0804, Train Accuracy: 0.9780, Train F1: 0.9850, Valid
Loss: 0.4672, Valid Accuracy: 0.8846, Valid F1: 0.9244
Epoch: 69, Train Loss: 0.0590, Train Accuracy: 0.9799, Train F1: 0.9862, Valid
Loss: 0.3183, Valid Accuracy: 0.9103, Valid F1: 0.9391
Epoch: 70, Train Loss: 0.0237, Train Accuracy: 0.9927, Train F1: 0.9950, Valid
Loss: 0.4731, Valid Accuracy: 0.9103, Valid F1: 0.9402
Epoch: 71, Train Loss: 0.0470, Train Accuracy: 0.9817, Train F1: 0.9875, Valid
Loss: 0.3144, Valid Accuracy: 0.9359, Valid F1: 0.9573
Epoch: 72, Train Loss: 0.0218, Train Accuracy: 0.9927, Train F1: 0.9950, Valid
Loss: 0.5507, Valid Accuracy: 0.8974, Valid F1: 0.9322
Epoch: 73, Train Loss: 0.0280, Train Accuracy: 0.9927, Train F1: 0.9950, Valid
Loss: 0.3862, Valid Accuracy: 0.9103, Valid F1: 0.9402
Epoch: 74, Train Loss: 0.0121, Train Accuracy: 0.9963, Train F1: 0.9975, Valid
Loss: 0.4342, Valid Accuracy: 0.9231, Valid F1: 0.9492
Epoch: 75, Train Loss: 0.0052, Train Accuracy: 0.9982, Train F1: 0.9987, Valid
Loss: 0.6866, Valid Accuracy: 0.8974, Valid F1: 0.9322
Epoch: 76, Train Loss: 0.0491, Train Accuracy: 0.9817, Train F1: 0.9875, Valid
Loss: 0.6697, Valid Accuracy: 0.8718, Valid F1: 0.9180
Epoch: 77, Train Loss: 0.0314, Train Accuracy: 0.9890, Train F1: 0.9925, Valid
Loss: 0.3327, Valid Accuracy: 0.9103, Valid F1: 0.9402
Epoch: 78, Train Loss: 0.0790, Train Accuracy: 0.9780, Train F1: 0.9850, Valid
Loss: 0.4530, Valid Accuracy: 0.9103, Valid F1: 0.9402
Epoch: 79, Train Loss: 0.0214, Train Accuracy: 0.9908, Train F1: 0.9937, Valid
Loss: 0.3994, Valid Accuracy: 0.9231, Valid F1: 0.9492
Epoch: 80, Train Loss: 0.0431, Train Accuracy: 0.9835, Train F1: 0.9887, Valid
Loss: 0.3588, Valid Accuracy: 0.9103, Valid F1: 0.9381
Epoch: 81, Train Loss: 0.0333, Train Accuracy: 0.9817, Train F1: 0.9875, Valid
Loss: 0.3567, Valid Accuracy: 0.9103, Valid F1: 0.9381
Epoch: 82, Train Loss: 0.0468, Train Accuracy: 0.9872, Train F1: 0.9913, Valid
Loss: 0.4689, Valid Accuracy: 0.8974, Valid F1: 0.9298
Epoch: 83, Train Loss: 0.0271, Train Accuracy: 0.9908, Train F1: 0.9937, Valid
Loss: 0.4376, Valid Accuracy: 0.9103, Valid F1: 0.9412
Epoch: 84, Train Loss: 0.0305, Train Accuracy: 0.9872, Train F1: 0.9912, Valid
```

```
Loss: 0.3809, Valid Accuracy: 0.9231, Valid F1: 0.9492
Epoch: 85, Train Loss: 0.0192, Train Accuracy: 0.9982, Train F1: 0.9987, Valid
Loss: 0.5053, Valid Accuracy: 0.9103, Valid F1: 0.9402
Epoch: 86, Train Loss: 0.0436, Train Accuracy: 0.9872, Train F1: 0.9912, Valid
Loss: 0.4953, Valid Accuracy: 0.8846, Valid F1: 0.9204
Epoch: 87, Train Loss: 0.0211, Train Accuracy: 0.9927, Train F1: 0.9950, Valid
Loss: 0.5600, Valid Accuracy: 0.8974, Valid F1: 0.9310
Epoch: 88, Train Loss: 0.0153, Train Accuracy: 0.9927, Train F1: 0.9950, Valid
Loss: 0.5929, Valid Accuracy: 0.9103, Valid F1: 0.9381
Epoch: 89, Train Loss: 0.1232, Train Accuracy: 0.9579, Train F1: 0.9709, Valid
Loss: 0.5557, Valid Accuracy: 0.8718, Valid F1: 0.9167
Epoch: 90, Train Loss: 0.0362, Train Accuracy: 0.9853, Train F1: 0.9900, Valid
Loss: 0.4540, Valid Accuracy: 0.8846, Valid F1: 0.9231
Epoch: 91, Train Loss: 0.0399, Train Accuracy: 0.9890, Train F1: 0.9925, Valid
Loss: 0.5018, Valid Accuracy: 0.8718, Valid F1: 0.9153
Epoch: 92, Train Loss: 0.0170, Train Accuracy: 0.9945, Train F1: 0.9962, Valid
Loss: 0.4615, Valid Accuracy: 0.9103, Valid F1: 0.9412
Epoch: 93, Train Loss: 0.0217, Train Accuracy: 0.9945, Train F1: 0.9962, Valid
Loss: 0.6137, Valid Accuracy: 0.8846, Valid F1: 0.9244
Epoch: 94, Train Loss: 0.0320, Train Accuracy: 0.9853, Train F1: 0.9900, Valid
Loss: 0.4884, Valid Accuracy: 0.9359, Valid F1: 0.9580
Epoch: 95, Train Loss: 0.0249, Train Accuracy: 0.9927, Train F1: 0.9950, Valid
Loss: 0.3753, Valid Accuracy: 0.8846, Valid F1: 0.9217
Epoch: 96, Train Loss: 0.0194, Train Accuracy: 0.9945, Train F1: 0.9962, Valid
Loss: 0.4054, Valid Accuracy: 0.8974, Valid F1: 0.9322
Epoch: 97, Train Loss: 0.0105, Train Accuracy: 0.9982, Train F1: 0.9987, Valid
Loss: 0.4594, Valid Accuracy: 0.9231, Valid F1: 0.9500
Epoch: 98, Train Loss: 0.0053, Train Accuracy: 1.0000, Train F1: 1.0000, Valid
Loss: 0.5342, Valid Accuracy: 0.9231, Valid F1: 0.9500
Epoch: 99, Train Loss: 0.0149, Train Accuracy: 0.9963, Train F1: 0.9975, Valid
Loss: 0.5045, Valid Accuracy: 0.8974, Valid F1: 0.9310
Epoch: 100, Train Loss: 0.0114, Train Accuracy: 0.9963, Train F1: 0.9975, Valid
Loss: 0.5833, Valid Accuracy: 0.9103, Valid F1: 0.9402
```

[ ]:
```python
import matplotlib.pyplot as plt
import torch
import pandas as pd

def plot_performance(loss_train_hist, loss_valid_hist, acc_train_hist,␣
 ↪acc_valid_hist):
    # Ensure all tensors are on CPU and convert to lists of numbers
    loss_train_hist = [loss.cpu().item() if isinstance(loss, torch.Tensor) else␣
 ↪loss for loss in loss_train_hist]
    loss_valid_hist = [loss.cpu().item() if isinstance(loss, torch.Tensor) else␣
 ↪loss for loss in loss_valid_hist]
    acc_train_hist = [acc.cpu().item() if isinstance(acc, torch.Tensor) else␣
 ↪acc for acc in acc_train_hist]
```

```
    acc_valid_hist = [acc.cpu().item() if isinstance(acc, torch.Tensor) else␣
↪acc for acc in acc_valid_hist]

    # Create a DataFrame for easier plotting
    data = pd.DataFrame({
        'Epoch': range(1, len(loss_train_hist) + 1),
        'Train Loss': loss_train_hist,
        'Validation Loss': loss_valid_hist,
        'Train Accuracy': acc_train_hist,
        'Validation Accuracy': acc_valid_hist,
    })

    # Calculate the rolling mean (moving average) to smooth the data
    window_size = 5
    data['Train Loss'] = data['Train Loss'].rolling(window=window_size).mean()
    data['Validation Loss'] = data['Validation Loss'].
↪rolling(window=window_size).mean()
    data['Train Accuracy'] = data['Train Accuracy'].rolling(window=window_size).
↪mean()
    data['Validation Accuracy'] = data['Validation Accuracy'].
↪rolling(window=window_size).mean()

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.plot(data['Epoch'], data['Train Loss'], label='Train Loss')
    plt.plot(data['Epoch'], data['Validation Loss'], label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(data['Epoch'], data['Train Accuracy'], label='Train Accuracy')
    plt.plot(data['Epoch'], data['Validation Accuracy'], label='Validation␣
↪Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout()
    plt.show()
```

[ ]:
```
plot_performance(vanilla_train_history['loss_train'],␣
↪vanilla_train_history['loss_valid'], vanilla_train_history['acc_train'],␣
↪vanilla_train_history['acc_valid'])
```



Discussing the Limitations of Accuracy

While accuracy is a commonly used metric, it can be misleading, especially in cases of class imbalance. In such scenarios, a model might predict the majority class for all instances, resulting in high accuracy but poor model performance in practical terms.

Why F1-Score? The F1-score is a more robust metric in these cases as it balances precision and recall, providing a better measure of the classifier's performance, especially when the classes are imbalanced.

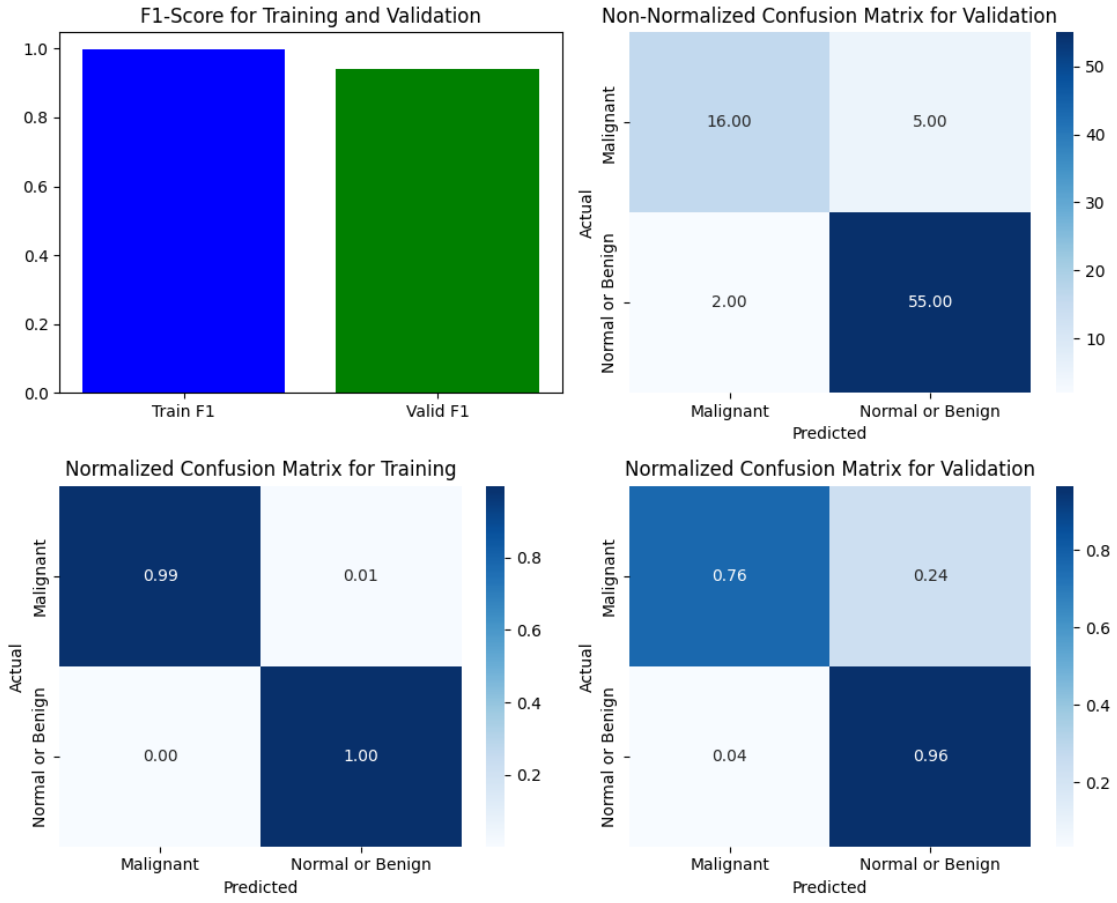
Importance of Confusion Matrix The confusion matrix provides an in-depth view of the classifier's performance. It shows not just the overall accuracy but how the model performs on each individual class, revealing any biases or weaknesses in the model's predictions.

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import torch
import numpy as np

def plot_f1_and_confusion_matrix(f1_train, f1_valid, cm_train, cm_valid):
    # Convert tensors to CPU and then to numbers if they are not already
    f1_train = f1_train.cpu().item() if isinstance(f1_train, torch.Tensor) else␣
 ↪f1_train
    f1_valid = f1_valid.cpu().item() if isinstance(f1_valid, torch.Tensor) else␣
 ↪f1_valid

    # Convert the confusion matrices to numpy arrays if they are tensors
    cm_train = cm_train.cpu().numpy() if isinstance(cm_train, torch.Tensor)␣
 ↪else cm_train
```

```
    cm_valid = cm_valid.cpu().numpy() if isinstance(cm_valid, torch.Tensor)␣
↪else cm_valid


    # Define a small constant to prevent division by zero (It is not needed for␣
↪correct implementation, provided just in case)
    eps = 1e-7

    # Normalize the confusion matrices
    cm_train_normalized = cm_train.astype('float') / (cm_train.sum(axis=1)[:,␣
↪np.newaxis] + eps)
    cm_valid_normalized = cm_valid.astype('float') / (cm_valid.sum(axis=1)[:,␣
↪np.newaxis] + eps)

    # Define the class labels
    class_labels = ['Malignant', 'Normal or Benign']

    # Plotting F1-Score
    plt.figure(figsize=(10, 8))
    plt.subplot(2, 2, 1)
    plt.bar(['Train F1', 'Valid F1'], [f1_train, f1_valid], color=['blue',␣
↪'green'])
    plt.title("F1-Score for Training and Validation")

    # Plotting Non-Normalized Confusion Matrix for Validation
    plt.subplot(2, 2, 2)
    sns.heatmap(cm_valid, annot=True, fmt='.2f', cmap='Blues',␣
↪xticklabels=class_labels, yticklabels=class_labels)
    plt.title("Non-Normalized Confusion Matrix for Validation")
    plt.ylabel('Actual')
    plt.xlabel('Predicted')

    # Plotting Normalized Confusion Matrix for Training
    plt.subplot(2, 2, 3)
    sns.heatmap(cm_train_normalized, annot=True, fmt='.2f', cmap='Blues',␣
↪xticklabels=class_labels, yticklabels=class_labels)
    plt.title("Normalized Confusion Matrix for Training")
    plt.ylabel('Actual')
    plt.xlabel('Predicted')

    # Plotting Normalized Confusion Matrix for Validation
    plt.subplot(2, 2, 4)
    sns.heatmap(cm_valid_normalized, annot=True, fmt='.2f', cmap='Blues',␣
↪xticklabels=class_labels, yticklabels=class_labels)
    plt.title("Normalized Confusion Matrix for Validation")
    plt.ylabel('Actual')
```

```
    plt.xlabel('Predicted')

    plt.tight_layout()
    plt.show()
```

[ ]:
```
plot_f1_and_confusion_matrix(vanilla_train_history['f1_train'][-1],␣
↪vanilla_train_history['f1_valid'][-1],␣
↪vanilla_train_history['cm_train'][-1], vanilla_train_history['cm_valid'][-1])
```



Importance of ROC and AUC in Evaluation:

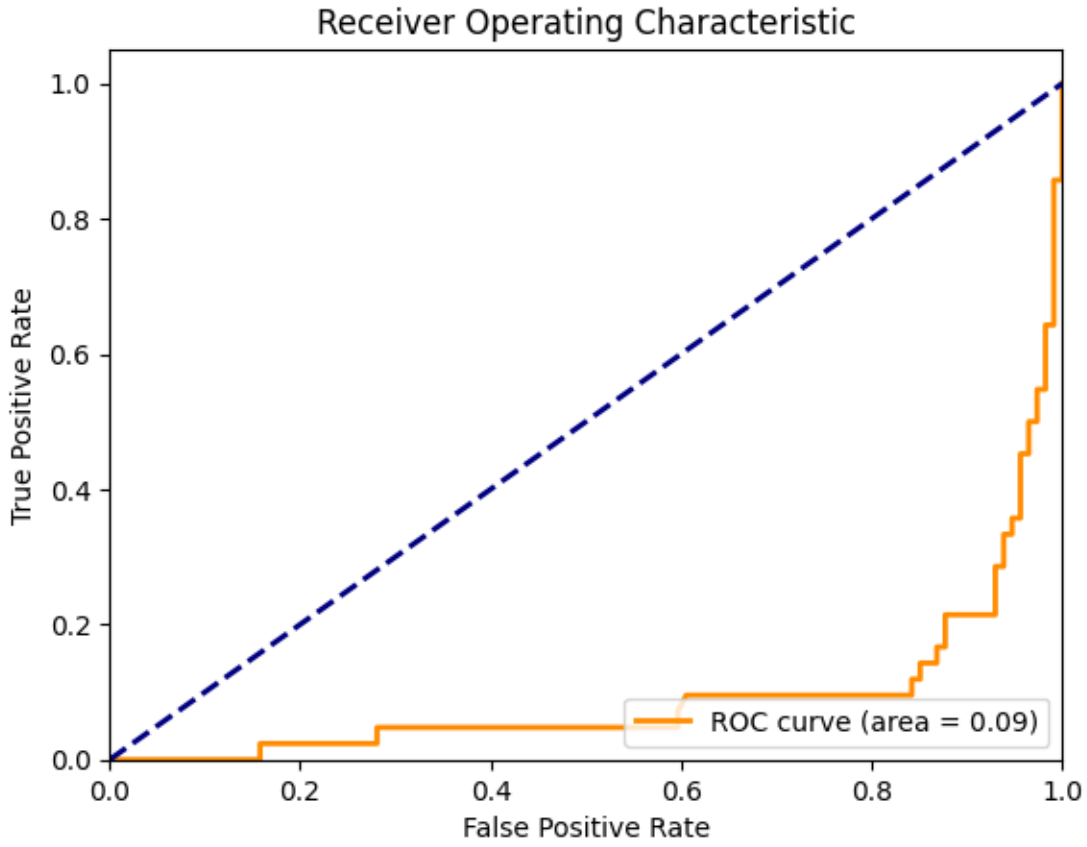
In medical imaging tasks like ours, accuracy isn't always the best performance metric due to potential class imbalances. Instead, we use ROC (Receiver Operating Characteristic) curves and AUC (Area Under the Curve) to provide a more nuanced view of our model's ability to distinguish between classes, ensuring a more reliable assessment of its diagnostic accuracy.

[ ]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import torch
```

```
def evaluate_and_plot_roc(model, data_loader, device):
    """
    Evaluates the model on the given data loader, calculates the ROC curve and␣
 ↪AUC,
    and plots the ROC curve.

    Parameters:
    model (torch.nn.Module): The trained model to evaluate.
    data_loader (torch.utils.data.DataLoader): DataLoader for evaluation data.
    device (torch.device): The device on which the model is.

    Returns:
    None
    """
    model.eval()  # Set the model to evaluation mode

    true_labels = []
    pred_probs = []

    with torch.no_grad():
        for data in data_loader:
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            probabilities = torch.sigmoid(outputs)

            # Store probabilities and true labels
            pred_probs.extend(probabilities.cpu().numpy())
            true_labels.extend(labels.cpu().numpy())

    # Invert the true labels: now 0 becomes 1 (malignant), and 1 becomes 0␣
 ↪(normal, benign)
    inverted_true_labels = [1 - label for label in true_labels]

    # Compute ROC curve and ROC area for each class
    fpr, tpr, _ = roc_curve(inverted_true_labels, pred_probs)
    roc_auc = auc(fpr, tpr)

    # Plotting ROC Curve
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange', lw=lw, label='ROC curve (area = %0.
 ↪2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
```

```
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc="lower right")
    plt.show()
```

[ ]:
```
import numpy as np
evaluate_and_plot_roc(model, test_loader, device)
```



2 Addressing Class Imbalance

Explanation

Class imbalance is a common challenge in medical data analysis. It happens when the number of examples in one class (usually the 'normal' category) is much larger than in another class (often representing a 'disease' condition). This imbalance can lead to models that are unfairly skewed towards the majority class, performing poorly in identifying the crucial, less represented class.

Why Address Class Imbalance?

In medical scenarios, the accuracy of detecting rare conditions (the minority class) is as important, if not more so, than identifying common ones. A model biased towards the majority class might overlook these critical minority cases, leading to potential misdiagnoses.

To counter this, one approach is to use a weighted loss function during training. Follow the following hints to implement it.

[ ]:
```
# Calculate class distribution
class_counts = np.bincount([label.item() for inputs, labels in train_loader for␣
 ↪label in labels])

# Calculate imbalance ratio
imbalance_ratio = class_counts.max() / class_counts

# Use the imbalance ratio of the 'malignant' class as the weight for the␣
 ↪positive class
pos_weight = torch.tensor([imbalance_ratio[0]], dtype=torch.float32).to(device)

# Define the loss function
criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)

print(f"Class distribution: {class_counts}")
print(f"Imbalance ratio: {imbalance_ratio}")
print(f"Weight for the positive ('malignant') class: {pos_weight.item()}")
```

```
Class distribution: [147 399]
Imbalance ratio: [2.71428571 1.        ]
Weight for the positive ('malignant') class: 2.7142856121063232
```

[ ]:
```
model = create_model(model_name="swin_tiny_patch4_window7_224",
                         pretrained=True,
                         num_classes=1,
                         in_chans=n_channels)
model = model.to(device)
```

[ ]:
```
from collections import defaultdict
from torch.optim import Adam
from torch.optim.lr_scheduler import ReduceLROnPlateau

n_epochs =  100
loss_fn_balanced = criterion

balanced_mode_history = train_model(model=model,␣
 ↪model_name='balanced_class_bcm_classifier', loss_fn=bce_loss,␣
 ↪optimizer=optimizer, device=device, metric='f1', n_epochs=n_epochs)
```

```
Best model saved. f1: 0.730434775352478
Epoch: 1, Train Loss: 0.6692, Train Accuracy: 0.6190, Train F1: 0.7326, Valid
Loss: 0.6589, Valid Accuracy: 0.6026, Valid F1: 0.7304
```

```
Epoch: 2, Train Loss: 0.6695, Train Accuracy: 0.6117, Train F1: 0.7310, Valid
Loss: 0.6558, Valid Accuracy: 0.6026, Valid F1: 0.7207
Best model saved. f1: 0.7543859481811523
Epoch: 3, Train Loss: 0.6633, Train Accuracy: 0.6172, Train F1: 0.7338, Valid
Loss: 0.6603, Valid Accuracy: 0.6410, Valid F1: 0.7544
Epoch: 4, Train Loss: 0.6662, Train Accuracy: 0.6154, Train F1: 0.7348, Valid
Loss: 0.6636, Valid Accuracy: 0.6282, Valid F1: 0.7478
Best model saved. f1: 0.7692307829856873
Epoch: 5, Train Loss: 0.6664, Train Accuracy: 0.6392, Train F1: 0.7471, Valid
Loss: 0.6517, Valid Accuracy: 0.6538, Valid F1: 0.7692
Epoch: 6, Train Loss: 0.6654, Train Accuracy: 0.6209, Train F1: 0.7396, Valid
Loss: 0.6495, Valid Accuracy: 0.6410, Valid F1: 0.7586
Epoch: 7, Train Loss: 0.6652, Train Accuracy: 0.6245, Train F1: 0.7348, Valid
Loss: 0.6539, Valid Accuracy: 0.6410, Valid F1: 0.7500
Epoch: 8, Train Loss: 0.6691, Train Accuracy: 0.6337, Train F1: 0.7442, Valid
Loss: 0.6566, Valid Accuracy: 0.6282, Valid F1: 0.7434
Best model saved. f1: 0.7719298005104065
Epoch: 9, Train Loss: 0.6688, Train Accuracy: 0.6062, Train F1: 0.7219, Valid
Loss: 0.6522, Valid Accuracy: 0.6667, Valid F1: 0.7719
Epoch: 10, Train Loss: 0.6728, Train Accuracy: 0.6117, Train F1: 0.7303, Valid
Loss: 0.6539, Valid Accuracy: 0.6538, Valid F1: 0.7568
Epoch: 11, Train Loss: 0.6651, Train Accuracy: 0.6484, Train F1: 0.7618, Valid
Loss: 0.6550, Valid Accuracy: 0.6026, Valid F1: 0.7304
Epoch: 12, Train Loss: 0.6671, Train Accuracy: 0.6026, Train F1: 0.7200, Valid
Loss: 0.6557, Valid Accuracy: 0.6538, Valid F1: 0.7652
Epoch: 13, Train Loss: 0.6703, Train Accuracy: 0.6026, Train F1: 0.7236, Valid
Loss: 0.6549, Valid Accuracy: 0.6667, Valid F1: 0.7719
Epoch: 14, Train Loss: 0.6638, Train Accuracy: 0.6484, Train F1: 0.7588, Valid
Loss: 0.6488, Valid Accuracy: 0.6538, Valid F1: 0.7652
Epoch: 15, Train Loss: 0.6651, Train Accuracy: 0.6264, Train F1: 0.7405, Valid
Loss: 0.6418, Valid Accuracy: 0.6538, Valid F1: 0.7692
Epoch: 16, Train Loss: 0.6647, Train Accuracy: 0.6502, Train F1: 0.7579, Valid
Loss: 0.6546, Valid Accuracy: 0.6410, Valid F1: 0.7544
Epoch: 17, Train Loss: 0.6709, Train Accuracy: 0.6136, Train F1: 0.7291, Valid
Loss: 0.6596, Valid Accuracy: 0.6282, Valid F1: 0.7521
Epoch: 18, Train Loss: 0.6724, Train Accuracy: 0.6026, Train F1: 0.7214, Valid
Loss: 0.6477, Valid Accuracy: 0.6667, Valid F1: 0.7636
Best model saved. f1: 0.7833333611488342
Epoch: 19, Train Loss: 0.6695, Train Accuracy: 0.6136, Train F1: 0.7298, Valid
Loss: 0.6554, Valid Accuracy: 0.6667, Valid F1: 0.7833
Epoch: 20, Train Loss: 0.6649, Train Accuracy: 0.6429, Train F1: 0.7516, Valid
Loss: 0.6536, Valid Accuracy: 0.6410, Valid F1: 0.7500
Epoch: 21, Train Loss: 0.6653, Train Accuracy: 0.6557, Train F1: 0.7626, Valid
Loss: 0.6603, Valid Accuracy: 0.6410, Valid F1: 0.7627
Best model saved. f1: 0.7894737124443054
Epoch: 22, Train Loss: 0.6678, Train Accuracy: 0.6282, Train F1: 0.7387, Valid
Loss: 0.6497, Valid Accuracy: 0.6923, Valid F1: 0.7895
Epoch: 23, Train Loss: 0.6636, Train Accuracy: 0.6337, Train F1: 0.7462, Valid
```

```
Loss: 0.6559, Valid Accuracy: 0.6282, Valid F1: 0.7521
Epoch: 24, Train Loss: 0.6685, Train Accuracy: 0.6099, Train F1: 0.7259, Valid
Loss: 0.6643, Valid Accuracy: 0.6154, Valid F1: 0.7321
Epoch: 25, Train Loss: 0.6705, Train Accuracy: 0.6227, Train F1: 0.7345, Valid
Loss: 0.6623, Valid Accuracy: 0.6026, Valid F1: 0.7207
Epoch: 26, Train Loss: 0.6687, Train Accuracy: 0.6044, Train F1: 0.7216, Valid
Loss: 0.6589, Valid Accuracy: 0.6154, Valid F1: 0.7368
Epoch: 27, Train Loss: 0.6719, Train Accuracy: 0.6044, Train F1: 0.7273, Valid
Loss: 0.6524, Valid Accuracy: 0.6538, Valid F1: 0.7731
Epoch: 28, Train Loss: 0.6671, Train Accuracy: 0.6117, Train F1: 0.7268, Valid
Loss: 0.6516, Valid Accuracy: 0.6410, Valid F1: 0.7544
Epoch: 29, Train Loss: 0.6719, Train Accuracy: 0.5952, Train F1: 0.7170, Valid
Loss: 0.6664, Valid Accuracy: 0.6282, Valid F1: 0.7478
Epoch: 30, Train Loss: 0.6685, Train Accuracy: 0.6429, Train F1: 0.7477, Valid
Loss: 0.6595, Valid Accuracy: 0.6026, Valid F1: 0.7207
Epoch: 31, Train Loss: 0.6687, Train Accuracy: 0.6117, Train F1: 0.7303, Valid
Loss: 0.6591, Valid Accuracy: 0.6410, Valid F1: 0.7544
Epoch: 32, Train Loss: 0.6659, Train Accuracy: 0.6190, Train F1: 0.7347, Valid
Loss: 0.6551, Valid Accuracy: 0.6154, Valid F1: 0.7458
Epoch: 33, Train Loss: 0.6695, Train Accuracy: 0.6245, Train F1: 0.7421, Valid
Loss: 0.6599, Valid Accuracy: 0.6282, Valid F1: 0.7434
Epoch: 34, Train Loss: 0.6659, Train Accuracy: 0.6355, Train F1: 0.7458, Valid
Loss: 0.6553, Valid Accuracy: 0.6282, Valid F1: 0.7603
Epoch: 35, Train Loss: 0.6657, Train Accuracy: 0.6319, Train F1: 0.7400, Valid
Loss: 0.6477, Valid Accuracy: 0.6410, Valid F1: 0.7627
Epoch: 36, Train Loss: 0.6672, Train Accuracy: 0.6190, Train F1: 0.7354, Valid
Loss: 0.6580, Valid Accuracy: 0.6795, Valid F1: 0.7863
Epoch: 37, Train Loss: 0.6762, Train Accuracy: 0.5934, Train F1: 0.7176, Valid
Loss: 0.6471, Valid Accuracy: 0.6538, Valid F1: 0.7692
Epoch: 38, Train Loss: 0.6683, Train Accuracy: 0.6245, Train F1: 0.7421, Valid
Loss: 0.6586, Valid Accuracy: 0.6282, Valid F1: 0.7387
Epoch: 39, Train Loss: 0.6658, Train Accuracy: 0.6374, Train F1: 0.7462, Valid
Loss: 0.6620, Valid Accuracy: 0.5513, Valid F1: 0.6847
Epoch: 40, Train Loss: 0.6679, Train Accuracy: 0.6136, Train F1: 0.7326, Valid
Loss: 0.6635, Valid Accuracy: 0.5897, Valid F1: 0.7241
Epoch: 41, Train Loss: 0.6668, Train Accuracy: 0.6465, Train F1: 0.7554, Valid
Loss: 0.6554, Valid Accuracy: 0.6154, Valid F1: 0.7458
Epoch: 42, Train Loss: 0.6690, Train Accuracy: 0.6319, Train F1: 0.7452, Valid
Loss: 0.6513, Valid Accuracy: 0.6667, Valid F1: 0.7797
Epoch: 43, Train Loss: 0.6665, Train Accuracy: 0.6355, Train F1: 0.7452, Valid
Loss: 0.6482, Valid Accuracy: 0.6667, Valid F1: 0.7719
Epoch: 44, Train Loss: 0.6681, Train Accuracy: 0.6099, Train F1: 0.7245, Valid
Loss: 0.6510, Valid Accuracy: 0.6923, Valid F1: 0.7895
Epoch: 45, Train Loss: 0.6721, Train Accuracy: 0.5751, Train F1: 0.6971, Valid
Loss: 0.6584, Valid Accuracy: 0.6154, Valid F1: 0.7414
Epoch: 46, Train Loss: 0.6647, Train Accuracy: 0.6190, Train F1: 0.7340, Valid
Loss: 0.6543, Valid Accuracy: 0.6410, Valid F1: 0.7586
Epoch: 47, Train Loss: 0.6653, Train Accuracy: 0.6575, Train F1: 0.7606, Valid
```

```
Loss: 0.6642, Valid Accuracy: 0.6410, Valid F1: 0.7586
Epoch: 48, Train Loss: 0.6669, Train Accuracy: 0.6484, Train F1: 0.7582, Valid
Loss: 0.6613, Valid Accuracy: 0.6282, Valid F1: 0.7387
Epoch: 49, Train Loss: 0.6696, Train Accuracy: 0.6227, Train F1: 0.7386, Valid
Loss: 0.6500, Valid Accuracy: 0.6795, Valid F1: 0.7863
Epoch: 50, Train Loss: 0.6637, Train Accuracy: 0.6081, Train F1: 0.7228, Valid
Loss: 0.6624, Valid Accuracy: 0.6026, Valid F1: 0.7257
Epoch: 51, Train Loss: 0.6700, Train Accuracy: 0.6245, Train F1: 0.7395, Valid
Loss: 0.6637, Valid Accuracy: 0.5769, Valid F1: 0.7130
Epoch: 52, Train Loss: 0.6672, Train Accuracy: 0.6337, Train F1: 0.7475, Valid
Loss: 0.6611, Valid Accuracy: 0.6026, Valid F1: 0.7350
Epoch: 53, Train Loss: 0.6594, Train Accuracy: 0.6410, Train F1: 0.7468, Valid
Loss: 0.6486, Valid Accuracy: 0.6410, Valid F1: 0.7500
Epoch: 54, Train Loss: 0.6672, Train Accuracy: 0.6374, Train F1: 0.7494, Valid
Loss: 0.6599, Valid Accuracy: 0.6026, Valid F1: 0.7304
Epoch: 55, Train Loss: 0.6718, Train Accuracy: 0.6264, Train F1: 0.7371, Valid
Loss: 0.6576, Valid Accuracy: 0.5769, Valid F1: 0.7179
Epoch: 56, Train Loss: 0.6666, Train Accuracy: 0.6209, Train F1: 0.7403, Valid
Loss: 0.6611, Valid Accuracy: 0.6026, Valid F1: 0.7395
Epoch: 57, Train Loss: 0.6713, Train Accuracy: 0.5897, Train F1: 0.7150, Valid
Loss: 0.6624, Valid Accuracy: 0.6154, Valid F1: 0.7273
Epoch: 58, Train Loss: 0.6679, Train Accuracy: 0.6209, Train F1: 0.7416, Valid
Loss: 0.6610, Valid Accuracy: 0.6282, Valid F1: 0.7478
Epoch: 59, Train Loss: 0.6635, Train Accuracy: 0.6355, Train F1: 0.7458, Valid
Loss: 0.6609, Valid Accuracy: 0.6538, Valid F1: 0.7652
Best model saved. f1: 0.8034188151359558
Epoch: 60, Train Loss: 0.6685, Train Accuracy: 0.6117, Train F1: 0.7350, Valid
Loss: 0.6438, Valid Accuracy: 0.7051, Valid F1: 0.8034
Epoch: 61, Train Loss: 0.6656, Train Accuracy: 0.6300, Train F1: 0.7462, Valid
Loss: 0.6532, Valid Accuracy: 0.6282, Valid F1: 0.7521
Epoch: 62, Train Loss: 0.6716, Train Accuracy: 0.6136, Train F1: 0.7305, Valid
Loss: 0.6640, Valid Accuracy: 0.5769, Valid F1: 0.7080
Epoch: 63, Train Loss: 0.6675, Train Accuracy: 0.6190, Train F1: 0.7347, Valid
Loss: 0.6592, Valid Accuracy: 0.6538, Valid F1: 0.7652
Epoch: 64, Train Loss: 0.6668, Train Accuracy: 0.6392, Train F1: 0.7528, Valid
Loss: 0.6451, Valid Accuracy: 0.6795, Valid F1: 0.7826
Epoch: 65, Train Loss: 0.6645, Train Accuracy: 0.6319, Train F1: 0.7478, Valid
Loss: 0.6559, Valid Accuracy: 0.6795, Valid F1: 0.7826
Epoch: 66, Train Loss: 0.6640, Train Accuracy: 0.6172, Train F1: 0.7344, Valid
Loss: 0.6558, Valid Accuracy: 0.6410, Valid F1: 0.7705
Epoch: 67, Train Loss: 0.6686, Train Accuracy: 0.6264, Train F1: 0.7411, Valid
Loss: 0.6577, Valid Accuracy: 0.6282, Valid F1: 0.7387
Best model saved. f1: 0.8099173307418823
Epoch: 68, Train Loss: 0.6675, Train Accuracy: 0.6209, Train F1: 0.7448, Valid
Loss: 0.6549, Valid Accuracy: 0.7051, Valid F1: 0.8099
Epoch: 69, Train Loss: 0.6629, Train Accuracy: 0.6319, Train F1: 0.7478, Valid
Loss: 0.6558, Valid Accuracy: 0.6154, Valid F1: 0.7414
Epoch: 70, Train Loss: 0.6658, Train Accuracy: 0.6319, Train F1: 0.7439, Valid
```

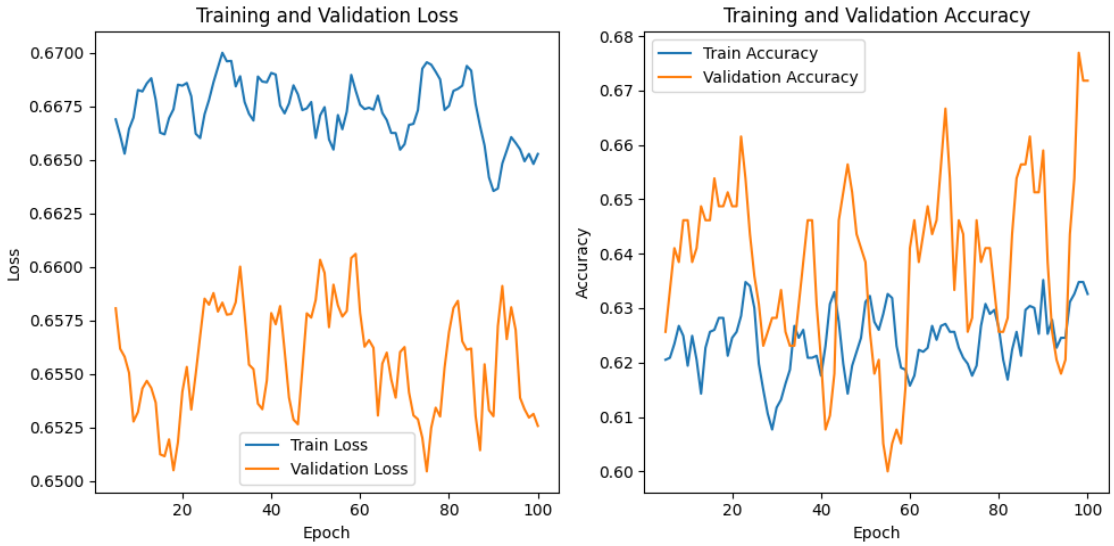
```
Loss: 0.6571, Valid Accuracy: 0.5769, Valid F1: 0.7130
Epoch: 71, Train Loss: 0.6685, Train Accuracy: 0.6026, Train F1: 0.7178, Valid
Loss: 0.6452, Valid Accuracy: 0.7051, Valid F1: 0.8000
Epoch: 72, Train Loss: 0.6689, Train Accuracy: 0.6172, Train F1: 0.7364, Valid
Loss: 0.6523, Valid Accuracy: 0.6154, Valid F1: 0.7368
Epoch: 73, Train Loss: 0.6707, Train Accuracy: 0.6154, Train F1: 0.7321, Valid
Loss: 0.6540, Valid Accuracy: 0.6154, Valid F1: 0.7414
Epoch: 74, Train Loss: 0.6726, Train Accuracy: 0.6209, Train F1: 0.7322, Valid
Loss: 0.6514, Valid Accuracy: 0.6282, Valid F1: 0.7563
Epoch: 75, Train Loss: 0.6672, Train Accuracy: 0.6410, Train F1: 0.7550, Valid
Loss: 0.6493, Valid Accuracy: 0.6667, Valid F1: 0.7797
Epoch: 76, Train Loss: 0.6679, Train Accuracy: 0.6392, Train F1: 0.7478, Valid
Loss: 0.6554, Valid Accuracy: 0.6667, Valid F1: 0.7797
Epoch: 77, Train Loss: 0.6672, Train Accuracy: 0.6374, Train F1: 0.7500, Valid
Loss: 0.6570, Valid Accuracy: 0.6282, Valid F1: 0.7521
Epoch: 78, Train Loss: 0.6689, Train Accuracy: 0.6062, Train F1: 0.7190, Valid
Loss: 0.6520, Valid Accuracy: 0.6154, Valid F1: 0.7414
Epoch: 79, Train Loss: 0.6654, Train Accuracy: 0.6245, Train F1: 0.7395, Valid
Loss: 0.6628, Valid Accuracy: 0.5897, Valid F1: 0.7241
Epoch: 80, Train Loss: 0.6682, Train Accuracy: 0.6245, Train F1: 0.7389, Valid
Loss: 0.6576, Valid Accuracy: 0.6282, Valid F1: 0.7521
Epoch: 81, Train Loss: 0.6714, Train Accuracy: 0.6099, Train F1: 0.7252, Valid
Loss: 0.6611, Valid Accuracy: 0.6667, Valid F1: 0.7759
Epoch: 82, Train Loss: 0.6677, Train Accuracy: 0.6190, Train F1: 0.7406, Valid
Loss: 0.6587, Valid Accuracy: 0.6410, Valid F1: 0.7544
Epoch: 83, Train Loss: 0.6696, Train Accuracy: 0.6337, Train F1: 0.7475, Valid
Loss: 0.6425, Valid Accuracy: 0.6923, Valid F1: 0.7931
Epoch: 84, Train Loss: 0.6701, Train Accuracy: 0.6410, Train F1: 0.7487, Valid
Loss: 0.6608, Valid Accuracy: 0.6410, Valid F1: 0.7705
Epoch: 85, Train Loss: 0.6671, Train Accuracy: 0.6026, Train F1: 0.7270, Valid
Loss: 0.6578, Valid Accuracy: 0.6410, Valid F1: 0.7544
Epoch: 86, Train Loss: 0.6636, Train Accuracy: 0.6520, Train F1: 0.7601, Valid
Loss: 0.6452, Valid Accuracy: 0.6667, Valid F1: 0.7636
Epoch: 87, Train Loss: 0.6626, Train Accuracy: 0.6227, Train F1: 0.7339, Valid
Loss: 0.6508, Valid Accuracy: 0.6667, Valid F1: 0.7797
Epoch: 88, Train Loss: 0.6651, Train Accuracy: 0.6319, Train F1: 0.7459, Valid
Loss: 0.6626, Valid Accuracy: 0.6410, Valid F1: 0.7667
Epoch: 89, Train Loss: 0.6626, Train Accuracy: 0.6172, Train F1: 0.7384, Valid
Loss: 0.6501, Valid Accuracy: 0.6410, Valid F1: 0.7586
Epoch: 90, Train Loss: 0.6639, Train Accuracy: 0.6520, Train F1: 0.7631, Valid
Loss: 0.6564, Valid Accuracy: 0.6795, Valid F1: 0.7788
Epoch: 91, Train Loss: 0.6642, Train Accuracy: 0.6026, Train F1: 0.7214, Valid
Loss: 0.6663, Valid Accuracy: 0.5641, Valid F1: 0.7018
Epoch: 92, Train Loss: 0.6684, Train Accuracy: 0.6355, Train F1: 0.7497, Valid
Loss: 0.6601, Valid Accuracy: 0.6026, Valid F1: 0.7257
Epoch: 93, Train Loss: 0.6681, Train Accuracy: 0.6062, Train F1: 0.7247, Valid
Loss: 0.6502, Valid Accuracy: 0.6154, Valid F1: 0.7500
Epoch: 94, Train Loss: 0.6658, Train Accuracy: 0.6264, Train F1: 0.7418, Valid
```

```
Loss: 0.6576, Valid Accuracy: 0.6282, Valid F1: 0.7603
Epoch: 95, Train Loss: 0.6626, Train Accuracy: 0.6520, Train F1: 0.7607, Valid
Loss: 0.6511, Valid Accuracy: 0.6923, Valid F1: 0.7966
Epoch: 96, Train Loss: 0.6626, Train Accuracy: 0.6355, Train F1: 0.7432, Valid
Loss: 0.6504, Valid Accuracy: 0.6795, Valid F1: 0.7826
Epoch: 97, Train Loss: 0.6657, Train Accuracy: 0.6429, Train F1: 0.7535, Valid
Loss: 0.6575, Valid Accuracy: 0.6538, Valid F1: 0.7692
Best model saved. f1: 0.8108108043670654
Epoch: 98, Train Loss: 0.6698, Train Accuracy: 0.6172, Train F1: 0.7338, Valid
Loss: 0.6482, Valid Accuracy: 0.7308, Valid F1: 0.8108
Epoch: 99, Train Loss: 0.6634, Train Accuracy: 0.6264, Train F1: 0.7398, Valid
Loss: 0.6584, Valid Accuracy: 0.6026, Valid F1: 0.7207
Epoch: 100, Train Loss: 0.6649, Train Accuracy: 0.6410, Train F1: 0.7494, Valid
Loss: 0.6483, Valid Accuracy: 0.6923, Valid F1: 0.7931
```

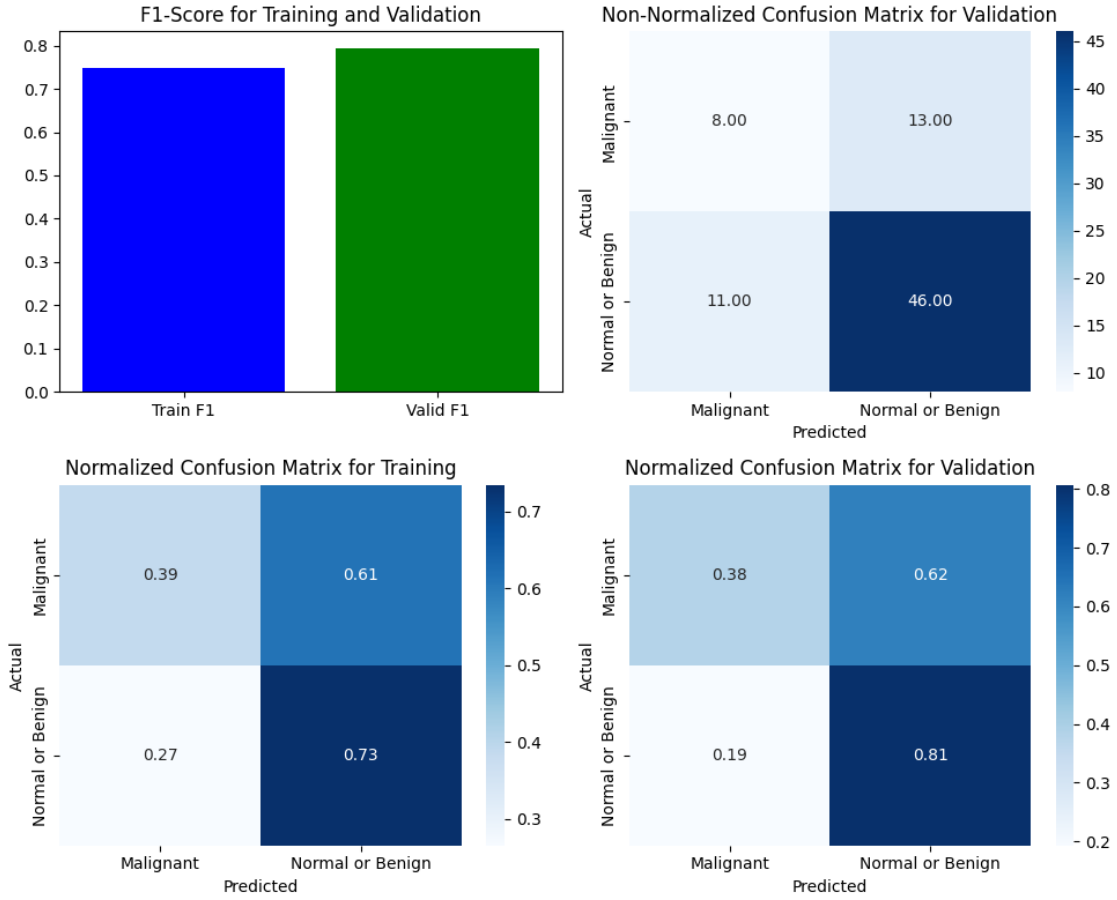
[ ]:
```
balanced_best_model, balanced_best_history = load_model(model,␣
↪'balanced_class_bcm_classifier')
```

```
Model loaded from ./models/balanced_class_bcm_classifier.pt
History loaded from ./models/balanced_class_bcm_classifier_history.pkl
```

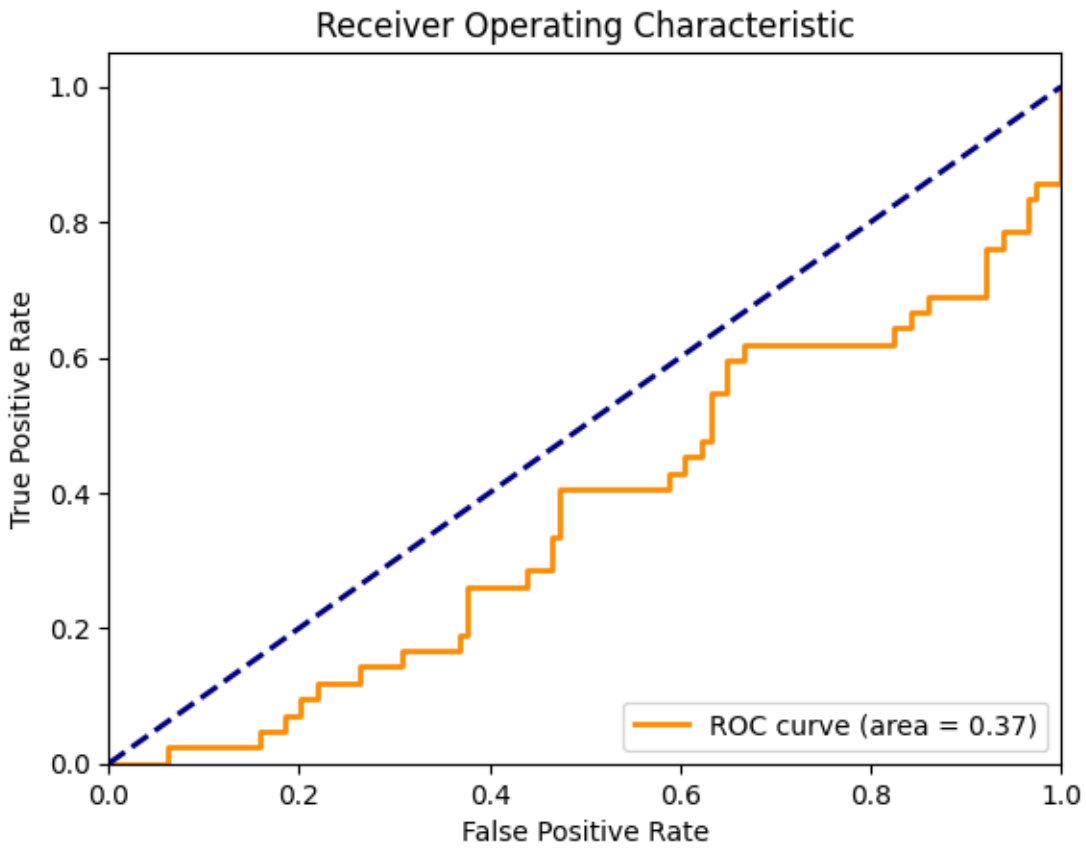
[ ]:
```
plot_performance(balanced_mode_history['loss_train'],␣
↪balanced_mode_history['loss_valid'], balanced_mode_history['acc_train'],␣
↪balanced_mode_history['acc_valid'])
```



[ ]:
```
plot_f1_and_confusion_matrix(balanced_mode_history['f1_train'][-1],␣
↪balanced_mode_history['f1_valid'][-1],␣
↪balanced_mode_history['cm_train'][-1], balanced_mode_history['cm_valid'][-1])
```



[ ]:
```
evaluate_and_plot_roc(model, test_loader, device)
```



In this notebook, you've already learned about addressing imbalanced data using a weighted loss. Now, let's explore an alternative method for tackling class imbalance in machine learning. Please describe and explain this alternative approach.

```
# Your answer here [15 score]

One alternative method for tackling class imbalance in machine learning is **sampling**. Sampli

There are different ways to implement sampling techniques, such as random sampling, synthetic s

Sampling techniques can be effective in reducing the class imbalance effect, but they also have

**Other approaches are**:

**Combination of Resampling and Ensemble Learning**: Another powerful approach is to combine re

**Anomaly Detection Techniques**: In some cases, treating the minority class instances as anoma

**Cost-sensitive Learning**: This approach involves modifying the learning algorithm itself to

**Customized Loss Functions**: Beyond standard weighted losses, developing custom loss functio
```

[ ]: