In [2]:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, models
from PIL import Image
import pandas as pd
import os
import numpy as np
from sklearn.model_selection import train_test_split
from tqdm import tqdm


def preprocess_image(image):
    image = image.convert("L")
    image = np.array(image)
    local_mean = np.mean(image)
    thresh_image = np.where(image > (local_mean - 2), 255, 0)
    return Image.fromarray(thresh_image.astype(np.uint8))


class GlomeruliDataset(Dataset):
    def __init__(self, csv_file, img_dir, transform=None):
        self.annotations = pd.read_csv(csv_file)
        self.img_dir = img_dir
        self.transform = transform

    def __len__(self):
        return len(self.annotations)

    def __getitem__(self, index):
        img_path = os.path.join(self.img_dir, self.annotations.iloc[index, 0])
        image = Image.open(img_path)
        image = preprocess_image(image)
        image = image.convert("RGB")
        y_label = torch.tensor(int(self.annotations.iloc[index, 1]))

        if self.transform:
            image = self.transform(image)
        return image, y_label


class GlomeruliClassifier(nn.Module):
    def __init__(self, base_model, num_classes=2):
        super(GlomeruliClassifier, self).__init__()
        self.base_model = base_model
        num_features = 2048
        self.classifier = nn.Sequential(
            nn.Linear(num_features, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes),
        )
    def forward(self, x):
        features = self.base_model(x)
        return self.classifier(features)


def train_model(
    model, train_loader, val_loader, criterion, optimizer, scheduler, num_epochs, d
):
    best_loss = float("inf")
    best_accuracy = 0.0

    for epoch in range(num_epochs):
        # Training phase
        model.train()
        running_loss = 0.0
        progress_bar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}")

        for images, labels in progress_bar:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

            progress_bar.set_postfix(
                {
                    "Loss": f"{running_loss/(progress_bar.n+1):.4f}",
                    "LR": f'{optimizer.param_groups[0]["lr"]:.6f}',
                }
            )

        epoch_loss = running_loss / len(train_loader)

        # Validation phase
        model.eval()
        val_loss = 0.0
        correct = 0
        total = 0

        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                _, predicted = outputs.max(1)
                total += labels.size(0)
                correct += predicted.eq(labels).sum().item()

        val_loss /= len(val_loader)
        val_accuracy = 100.0 * correct / total

        print(f"\nEpoch {epoch+1}/{num_epochs}")
        print(f"Average Loss: {epoch_loss:.4f}")
        print(f"Validation Loss: {val_loss:.4f}")
        print(f"Validation Accuracy: {val_accuracy:.2f}%")

        if val_loss < best_loss:
            best_loss = val_loss
            torch.save(model.state_dict(), "Final_model.pth")
            print(f"Saved new best model with validation loss: {best_loss:.4f}")

        scheduler.step(val_loss)
        print("\n")


if __name__ == "__main__":
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Using device: {device}")

    transform = transforms.Compose(
        [
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.2
        ]
    )

    dataset = GlomeruliDataset(
        csv_file="./train_labels.csv",
        img_dir="./ResizedTrainingSet",
        transform=transform,
    )

    labels = [int(dataset.annotations.iloc[i, 1]) for i in range(len(dataset))]
    class_counts = np.bincount(labels)
    total_samples = len(labels)
    class_weights = total_samples / (len(class_counts) * class_counts)
    class_weights = torch.FloatTensor(class_weights).to(device)
    print("Class distribution:", class_counts)
    print("Class weights:", class_weights.cpu().numpy())

    train_set, val_set = train_test_split(dataset, test_size=0.2, random_state=42)
    train_loader = DataLoader(train_set, batch_size=32, shuffle=True, num_workers=2
    val_loader = DataLoader(val_set, batch_size=32, shuffle=False, num_workers=2)

    model_path = "./inception_v3_google-0cc3c7bd.pth"
    base_model = models.inception_v3(pretrained=False)
    base_model.load_state_dict(torch.load(model_path))
    base_model.aux_logits = False
    base_model.fc = nn.Identity()
    model = GlomeruliClassifier(base_model).to(device)

    criterion = nn.CrossEntropyLoss(weight=class_weights)
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(
        optimizer, mode="min", factor=0.1, patience=3, verbose=True
    )

    train_model(
        model,
        train_loader,
        val_loader,
        criterion,
        optimizer,
        scheduler,
        num_epochs=20,
        device=device,
    )
```

```
Using device: cuda
Class distribution: [3763  843]
Class weights: [0.6120117 2.7319098]
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarnin
g: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the fut
ure, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarnin
g: 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=None`.
  warnings.warn(msg)
/usr/local/lib/python3.10/dist-packages/torchvision/models/inception.py:43: FutureWa
rning: The default weight initialization of inception_v3 will be changed in future r
eleases of torchvision. If you wish to keep the old behavior (which leads to long in
itialization times due to scipy/scipy#11299), please set init_weights=True.
  warnings.warn(
<ipython-input-2-86d37c2338d1>:155: FutureWarning: You are using `torch.load` with `
weights_only=False` (the current default value), which uses the default pickle modul
e implicitly. It is possible to construct malicious pickle data which will execute a
rbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SE
CURITY.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 b
e executed during unpickling. Arbitrary objects will no longer be allowed to be load
ed via this mode unless they are explicitly allowlisted by the user via `torch.seria
lization.add_safe_globals`. We recommend you start setting `weights_only=True` for a
ny use case where you don't have full control of the loaded file. Please open an iss
ue on GitHub for any issues related to this experimental feature.
  base_model.load_state_dict(torch.load(model_path))
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:60: UserWarning:
The verbose parameter is deprecated. Please use get_last_lr() to access the learning
rate.
  warnings.warn(
Epoch 1/20: 100%|██████████| 116/116 [00:50<00:00,  2.28it/s, Loss=0.3494, LR=0.0010
00]
```

```
Epoch 1/20
Average Loss: 0.3494
Validation Loss: 0.1550
Validation Accuracy: 93.17%
Saved new best model with validation loss: 0.1550
```

```
Epoch 2/20: 100%|██████████| 116/116 [00:52<00:00,  2.22it/s, Loss=0.2473, LR=0.0010
00]
```

```
Epoch 2/20
Average Loss: 0.2473
Validation Loss: 0.1519
Validation Accuracy: 93.82%
Saved new best model with validation loss: 0.1519
```

```
Epoch 3/20: 100%|██████████| 116/116 [00:53<00:00,  2.18it/s, Loss=0.1861, LR=0.0010
00]
```

```
Epoch 3/20
Average Loss: 0.1861
Validation Loss: 0.3580
Validation Accuracy: 80.69%
```

```
Epoch 4/20: 100%|██████████| 116/116 [00:54<00:00,  2.15it/s, Loss=0.1221, LR=0.0010
00]
```

```
Epoch 4/20
Average Loss: 0.1221
Validation Loss: 0.2450
Validation Accuracy: 93.28%
```

```
Epoch 5/20: 100%|██████████| 116/116 [00:54<00:00,  2.12it/s, Loss=0.1136, LR=0.0010
00]
```

```
Epoch 5/20
Average Loss: 0.1136
Validation Loss: 0.2634
Validation Accuracy: 94.58%
```

```
Epoch 6/20: 100%|██████████| 116/116 [00:55<00:00,  2.11it/s, Loss=0.1631, LR=0.0010
00]
```

```
Epoch 6/20
Average Loss: 0.1631
Validation Loss: 0.2793
Validation Accuracy: 87.96%
```

```
Epoch 7/20: 100%|██████████| 116/116 [00:55<00:00,  2.09it/s, Loss=0.0729, LR=0.0001
00]
```

```
Epoch 7/20
Average Loss: 0.0729
Validation Loss: 0.1180
Validation Accuracy: 96.10%
Saved new best model with validation loss: 0.1180
```

```
Epoch 8/20: 100%|██████████| 116/116 [00:55<00:00,  2.10it/s, Loss=0.0490, LR=0.0001
00]
```

```
Epoch 8/20
Average Loss: 0.0490
Validation Loss: 0.1560
Validation Accuracy: 96.20%
```

```
Epoch 9/20: 100%|██████████| 116/116 [00:55<00:00,  2.10it/s, Loss=0.0348, LR=0.0001
00]
Epoch 9/20
Average Loss: 0.0348
Validation Loss: 0.1573
Validation Accuracy: 96.10%
```

```
Epoch 10/20: 100%|██████████| 116/116 [00:55<00:00,  2.09it/s, Loss=0.0263, LR=0.000
100]
Epoch 10/20
Average Loss: 0.0263
Validation Loss: 0.1559
Validation Accuracy: 96.10%
```

```
Epoch 11/20: 100%|██████████| 116/116 [00:55<00:00,  2.09it/s, Loss=0.0291, LR=0.000
100]
Epoch 11/20
Average Loss: 0.0291
Validation Loss: 0.1931
Validation Accuracy: 96.20%
```

```
Epoch 12/20: 100%|██████████| 116/116 [00:55<00:00,  2.09it/s, Loss=0.0138, LR=0.000
010]
Epoch 12/20
Average Loss: 0.0138
Validation Loss: 0.1798
Validation Accuracy: 95.44%
```

```
Epoch 13/20: 100%|██████████| 116/116 [00:55<00:00,  2.09it/s, Loss=0.0096, LR=0.000
010]
Epoch 13/20
Average Loss: 0.0096
Validation Loss: 0.2013
Validation Accuracy: 95.66%
```

```
Epoch 14/20: 100%|██████████| 116/116 [00:55<00:00,  2.08it/s, Loss=0.0193, LR=0.000
010]
Epoch 14/20
Average Loss: 0.0193
Validation Loss: 0.2241
Validation Accuracy: 96.42%
```

```
Epoch 15/20: 100%|██████████| 116/116 [00:55<00:00,  2.09it/s, Loss=0.0118, LR=0.000
010]
Epoch 15/20
Average Loss: 0.0118
Validation Loss: 0.1762
Validation Accuracy: 95.66%


Epoch 16/20: 100%|██████████| 116/116 [00:55<00:00,  2.09it/s, Loss=0.0098, LR=0.000
001]
Epoch 16/20
Average Loss: 0.0098
Validation Loss: 0.1782
Validation Accuracy: 95.99%


Epoch 17/20: 100%|██████████| 116/116 [00:55<00:00,  2.09it/s, Loss=0.0178, LR=0.000
001]
Epoch 17/20
Average Loss: 0.0178
Validation Loss: 0.1903
Validation Accuracy: 96.10%


Epoch 18/20: 100%|██████████| 116/116 [00:55<00:00,  2.09it/s, Loss=0.0131, LR=0.000
001]
Epoch 18/20
Average Loss: 0.0131
Validation Loss: 0.1878
Validation Accuracy: 95.66%


Epoch 19/20: 100%|██████████| 116/116 [00:55<00:00,  2.09it/s, Loss=0.0125, LR=0.000
001]
Epoch 19/20
Average Loss: 0.0125
Validation Loss: 0.1884
Validation Accuracy: 95.77%


Epoch 20/20: 100%|██████████| 116/116 [00:55<00:00,  2.09it/s, Loss=0.0111, LR=0.000
000]
Epoch 20/20
Average Loss: 0.0111
Validation Loss: 0.1644
Validation Accuracy: 95.99%
```

In [5]:

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, models
from PIL import Image
```

```python
import pandas as pd
import os
import numpy as np
from tqdm import tqdm
from sklearn.metrics import classification_report, confusion_matrix


def preprocess_image(image):
    image = image.convert("L")
    image = np.array(image)
    local_mean = np.mean(image)
    thresh_image = np.where(image > (local_mean - 2), 255, 0)
    return Image.fromarray(thresh_image.astype(np.uint8))


class GlomeruliDataset(Dataset):
    def __init__(self, csv_file, img_dir, transform=None):
        self.annotations = pd.read_csv(csv_file)
        self.img_dir = img_dir
        self.transform = transform

    def __len__(self):
        return len(self.annotations)

    def __getitem__(self, index):
        img_path = os.path.join(self.img_dir, self.annotations.iloc[index, 0])
        image = Image.open(img_path)
        image = preprocess_image(image)
        image = image.convert("RGB")
        y_label = torch.tensor(int(self.annotations.iloc[index, 1]))
        if self.transform:
            image = self.transform(image)
        return image, y_label


class GlomeruliClassifier(nn.Module):
    def __init__(self, base_model, num_classes=2):
        super(GlomeruliClassifier, self).__init__()
        self.base_model = base_model
        num_features = 2048
        self.classifier = nn.Sequential(
            nn.Linear(num_features, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes),
        )

    def forward(self, x):
        features = self.base_model(x)
        return self.classifier(features)


def evaluate_model(model, test_loader, device):
    model.eval()
    correct = 0
    total = 0
    predictions = []
    true_labels = []

    with torch.no_grad():
        for images, labels in tqdm(test_loader):
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = outputs.max(1)

            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

            predictions.extend(predicted.cpu().numpy())
            true_labels.extend(labels.cpu().numpy())

    accuracy = 100.0 * correct / total
    return accuracy, predictions, true_labels


if __name__ == "__main__":
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Using device: {device}")

    # Data transformations
    transform = transforms.Compose(
        [
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.2
        ]
    )

    # Test data
    test_dataset = GlomeruliDataset(
        csv_file="./test_labels.csv",  # Path to your test labels CSV
        img_dir="./ResizedTestSet",  # Path to your test images directory
        transform=transform,
    )
    test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, num_worker

    # Create a models directory in your project
    model_path = "./inception_v3_google-0cc3c7bd.pth"

    # Load the model with custom weights using safe loading
    base_model = models.inception_v3(weights=None, init_weights=True)
    base_model.load_state_dict(torch.load(model_path, weights_only=True))
    base_model.fc = nn.Identity()
    model = GlomeruliClassifier(base_model).to(device)

    # Load the trained model
    model.load_state_dict(torch.load("./Final_model.pth"))

    # Evaluate the model
    accuracy, predictions, true_labels = evaluate_model(model, test_loader, device)

    print(f"\nTest Accuracy: {accuracy:.2f}%")
    print("\nClassification Report:")
    print(classification_report(true_labels, predictions))
    print("\nConfusion Matrix:")
    print(confusion_matrix(true_labels, predictions))
```

```
Using device: cuda
<ipython-input-5-9b37f231db4e>:112: FutureWarning: You are using `torch.load` with `
weights_only=False` (the current default value), which uses the default pickle modul
e implicitly. It is possible to construct malicious pickle data which will execute a
rbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SE
CURITY.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 b
e executed during unpickling. Arbitrary objects will no longer be allowed to be load
ed via this mode unless they are explicitly allowlisted by the user via `torch.seria
lization.add_safe_globals`. We recommend you start setting `weights_only=True` for a
ny use case where you don't have full control of the loaded file. Please open an iss
ue on GitHub for any issues related to this experimental feature.
  model.load_state_dict(torch.load("/kaggle/working/Final_model.pth"))
100%|██████████| 36/36 [00:06<00:00,  5.55it/s]
Test Accuracy: 95.75%

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.95      0.97       941
           1       0.83      0.97      0.89       211

    accuracy                           0.96      1152
   macro avg       0.91      0.96      0.93      1152
weighted avg       0.96      0.96      0.96      1152


Confusion Matrix:
[[898  43]
 [  6 205]]
```

In [3]:

```
import torch
import torch.nn as nn
from torchvision import transforms, models
from PIL import Image
import os
import numpy as np
import csv
from tqdm import tqdm

def preprocess_image(image):
    image = image.convert("L")
    image = np.array(image)
    local_mean = np.mean(image)
    thresh_image = np.where(image > (local_mean - 2), 255, 0)
    return Image.fromarray(thresh_image.astype(np.uint8))

class GlomeruliClassifier(nn.Module):
    def __init__(self, base_model, num_classes=2):
        super(GlomeruliClassifier, self).__init__()
        self.base_model = base_model
        num_features = 2048
        self.classifier = nn.Sequential(
            nn.Linear(num_features, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes),
        )

    def forward(self, x):
        features = self.base_model(x)
        return self.classifier(features)

def predict_images(model, image_folder, device):
    model.eval()
    predictions = []
    transform = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])

    with torch.no_grad():
        for image_name in tqdm(os.listdir(image_folder)):
            if image_name.lower().endswith(('.png', '.jpg', '.jpeg')):
                img_path = os.path.join(image_folder, image_name)
                image = Image.open(img_path)
                image = preprocess_image(image)
                image = image.convert("RGB")
                image = transform(image).unsqueeze(0).to(device)

                outputs = model(image)
                _, predicted = outputs.max(1)
                predictions.append((image_name, predicted.item()))

    return predictions

def save_predictions_to_csv(predictions, output_file):
    with open(output_file, 'w', newline='') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(['Image Name', 'Prediction'])
        writer.writerows(predictions)

if __name__ == "__main__":
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Using device: {device}")

    # Load the model
    model_path = "./inception_v3_google-0cc3c7bd.pth"
    base_model = models.inception_v3(weights=None, init_weights=True)
    base_model.load_state_dict(torch.load(model_path, weights_only=True))
    base_model.fc = nn.Identity()
    model = GlomeruliClassifier(base_model).to(device)

    # Load the trained model weights
    model.load_state_dict(torch.load("./Final_model.pth"))
    # Specify the folder containing the images to be classified
    image_folder = "./ResizedTestSet"

    # Predict classifications for all images in the folder
    predictions = predict_images(model, image_folder, device)

    # Save predictions to CSV
    output_file = "evaluation.csv"
    save_predictions_to_csv(predictions, output_file)

    print(f"Predictions saved to {output_file}")
```

```
Using device: cuda
<ipython-input-3-5b22aa7a7c31>:75: FutureWarning: You are using `torch.load` with `w
eights_only=False` (the current default value), which uses the default pickle module
implicitly. It is possible to construct malicious pickle data which will execute arb
itrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECU
RITY.md#untrusted-models for more details). In a future release, the default value f
or `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.serializ
ation.add_safe_globals`. We recommend you start setting `weights_only=True` for any
use case where you don't have full control of the loaded file. Please open an issue
on GitHub for any issues related to this experimental feature.
  model.load_state_dict(torch.load("/kaggle/working/Final_model.pth"))
100%|██████████| 1152/1152 [00:30<00:00, 37.65it/s]
Predictions saved to evaluation.csv
```