Convolutional Neural Network

Dataset in use: https://www.kaggle.com/datasets/cashutosh/gender-classification-dataset/data (https://www.kaggle.com/datasets/cashutosh/gender-classification-dataset/data)

In [1]:
```
import os
import torch
from PIL import Image
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import matplotlib.pyplot as plt
from torch.optim.lr_scheduler import StepLR
from torch.utils.data import DataLoader, Dataset
from torchvision import models, transforms, datasets
from sklearn.metrics import confusion_matrix, classification_report, Confusi
```

In [2]:
```
train_dir = "./dataset/Training"
val_dir = "./dataset/Validation"
input_size = (224, 224)
batch_size = 64
```

In [3]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

In [4]:
```
transform = {
    'train': transforms.Compose([
        transforms.Resize(input_size),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(input_size),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

In [5]:
```
image_datasets = {
    'train': datasets.ImageFolder(root=train_dir, transform=transform['train
    'val': datasets.ImageFolder(root=val_dir, transform=transform['val'])
}

dataloaders = {
    'train': DataLoader(image_datasets['train'], batch_size=batch_size, shuf
    'val': DataLoader(image_datasets['val'], batch_size=batch_size, shuffle=
}
```

In [6]:
```
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes
```

In [7]:
```
class GenderClassifier(nn.Module):
    def __init__(self):
        super(GenderClassifier, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(64)
        self.fc1 = nn.Linear(64 * 28 * 28, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.pool(self.bn1(nn.ReLU()(self.conv1(x))))
        x = self.pool(self.bn2(nn.ReLU()(self.conv2(x))))
        x = self.pool(self.bn3(nn.ReLU()(self.conv3(x))))
        x = x.view(-1, 64 * 28 * 28)
        x = nn.ReLU()(self.fc1(x))
        x = nn.ReLU()(self.fc2(x))
        x = self.sigmoid(self.fc3(x))
        return x
```

In [8]:
```
model = GenderClassifier()

model.to(device)
# Define loss function and optimizer
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters())
```

In [9]:
```
num_epochs = 3
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct_predictions = 0
    total_predictions = 0

    for batch_idx, (inputs, labels) in enumerate(dataloaders['train']):
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels.float().view(-1, 1))
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * inputs.size(0)

        # Calculate accuracy for this batch
        predicted = torch.round(outputs)
        correct_predictions += torch.sum(predicted == labels.view_as(predict
        total_predictions += labels.size(0)

        batch_loss = loss.item()
        print(f'Epoch [{epoch+1}/{num_epochs}], Phase: train, Batch: [{batch

    epoch_loss = running_loss / dataset_sizes['train']
    epoch_accuracy = correct_predictions / total_predictions
    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {epoch_loss:.4f}, Tr
```

```
Epoch [2/3], Phase: train, Batch: [370/730], Loss: 0.1810, Accuracy: 0.95
41
Epoch [2/3], Phase: train, Batch: [371/730], Loss: 0.1104, Accuracy: 0.95
41
Epoch [2/3], Phase: train, Batch: [372/730], Loss: 0.1837, Accuracy: 0.95
40
Epoch [2/3], Phase: train, Batch: [373/730], Loss: 0.2032, Accuracy: 0.95
39
Epoch [2/3], Phase: train, Batch: [374/730], Loss: 0.1679, Accuracy: 0.95
38
Epoch [2/3], Phase: train, Batch: [375/730], Loss: 0.1248, Accuracy: 0.95
38
Epoch [2/3], Phase: train, Batch: [376/730], Loss: 0.1572, Accuracy: 0.95
38
Epoch [2/3], Phase: train, Batch: [377/730], Loss: 0.1520, Accuracy: 0.95
37
Epoch [2/3], Phase: train, Batch: [378/730], Loss: 0.1382, Accuracy: 0.95
36
Epoch [2/3], Phase: train, Batch: [379/730], Loss: 0.1870, Accuracy: 0.95
35
```

In [10]:
```
true_labels = []
predicted_labels = []

model.eval()

with torch.no_grad():
    for batch_idx, (inputs, labels) in enumerate(dataloaders['val']):
        inputs, labels = inputs.to(device), labels.to(device)

        outputs = model(inputs)
        predicted = (outputs >= 0.5).squeeze().long()

        true_labels.extend(labels.cpu().numpy())
        predicted_labels.extend(predicted.cpu().numpy())
```

In [11]:
```
conf_matrix = confusion_matrix(true_labels, predicted_labels)
classification_rep = classification_report(true_labels, predicted_labels, ta

print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[5743   98]
 [ 461 5345]]
```

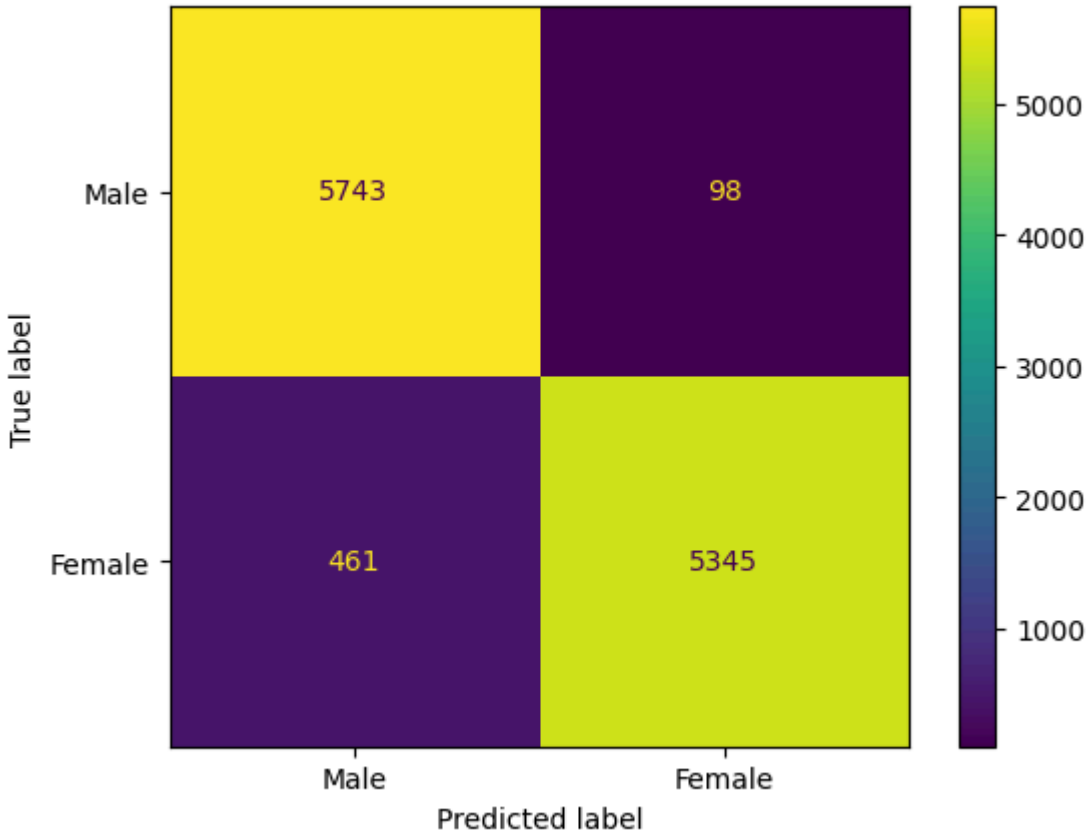
In [12]:
```
print("Classification Report:")
print(classification_rep)
```

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

      female       0.93      0.98      0.95      5841
        male       0.98      0.92      0.95      5806

    accuracy                           0.95     11647
   macro avg       0.95      0.95      0.95     11647
weighted avg       0.95      0.95      0.95     11647
```

In [13]:
```
cm_display = ConfusionMatrixDisplay(confusion_matrix = conf_matrix, display_
cm_display.plot()
plt.show()
```



In [14]:
```
torch.save(model.state_dict(), 'custom_model.pth')
```