Convolutional Neural Network

Dataset in use: https://susanqq.github.io/UTKFace/ (https://susanqq.github.io/UTKFace/)

In-the-wild Faces is used and part-2 is selected for train, part-3 is selected for test set.

In [21]:
```
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 torchvision import models, transforms
from torch.optim.lr_scheduler import StepLR
from torch.utils.data import DataLoader, Dataset
from sklearn.metrics import confusion_matrix, classification_report, ConfusionMatrixDisplay
```

In [2]:
```
train_dir = 'utk_train_cropped'
val_dir = 'utk_test_cropped'
input_size = (224, 224)
batch_size = 64
```

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

In [4]:
```
# Define your custom dataset
class GenderDataset(Dataset):
    def __init__(self, directory, transform=None):
        self.directory = directory
        self.transform = transform
        self.filenames = os.listdir(directory)

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

    def __getitem__(self, idx):
        img_name = self.filenames[idx]
        img_path = os.path.join(self.directory, img_name)
        image = Image.open(img_path)

        gender_label = int(img_name.split('_')[1])

        if self.transform:
            image = self.transform(image)

        return image, gender_label
```

In [5]:
```
transform = {
    'train': transforms.Compose([
        transforms.Resize(input_size),
        transforms.Grayscale(num_output_channels=3),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(input_size),
        transforms.Grayscale(num_output_channels=3),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

In [6]:
```
image_datasets = {
    'train': GenderDataset(directory=train_dir, transform=transform['train']),
    'val': GenderDataset(directory=val_dir, transform=transform['val'])
}

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

In [7]:
```
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
```

In [8]:
```
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 [9]:
```
model = GenderClassifier()

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

In [10]:
```
num_epochs = 10
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(predicted)).item()
        total_predictions += labels.size(0)

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

    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}, Train Accuracy: {epoch_accuracy:.4f}'
```

```
Epoch [2/10], Phase: train, Batch: [4/131], Loss: 0.2466
Epoch [2/10], Phase: train, Batch: [5/131], Loss: 0.4631
Epoch [2/10], Phase: train, Batch: [6/131], Loss: 0.3524
Epoch [2/10], Phase: train, Batch: [7/131], Loss: 0.4836
Epoch [2/10], Phase: train, Batch: [8/131], Loss: 0.2158
Epoch [2/10], Phase: train, Batch: [9/131], Loss: 0.2758
Epoch [2/10], Phase: train, Batch: [10/131], Loss: 0.3009

Epoch [2/10], Phase: train, Batch: [11/131], Loss: 0.3319
Epoch [2/10], Phase: train, Batch: [12/131], Loss: 0.1976
Epoch [2/10], Phase: train, Batch: [13/131], Loss: 0.3806
Epoch [2/10], Phase: train, Batch: [14/131], Loss: 0.2330
Epoch [2/10], Phase: train, Batch: [15/131], Loss: 0.1949
Epoch [2/10], Phase: train, Batch: [16/131], Loss: 0.2578
Epoch [2/10], Phase: train, Batch: [17/131], Loss: 0.2658
Epoch [2/10], Phase: train, Batch: [18/131], Loss: 0.2367
Epoch [2/10], Phase: train, Batch: [19/131], Loss: 0.3115
Epoch [2/10], Phase: train, Batch: [20/131], Loss: 0.4667
Epoch [2/10], Phase: train, Batch: [21/131], Loss: 0.2484
Epoch [2/10], Phase: train, Batch: [22/131], Loss: 0.4620
```

In [18]:
```
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 [16]:
```
class_names = ['male', 'female']
```

In [17]:
```
conf_matrix = confusion_matrix(true_labels, predicted_labels)
classification_rep = classification_report(true_labels, predicted_labels, target_names=class_names)

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

```
Confusion Matrix:
[[1871  105]
 [ 152  626]]
```

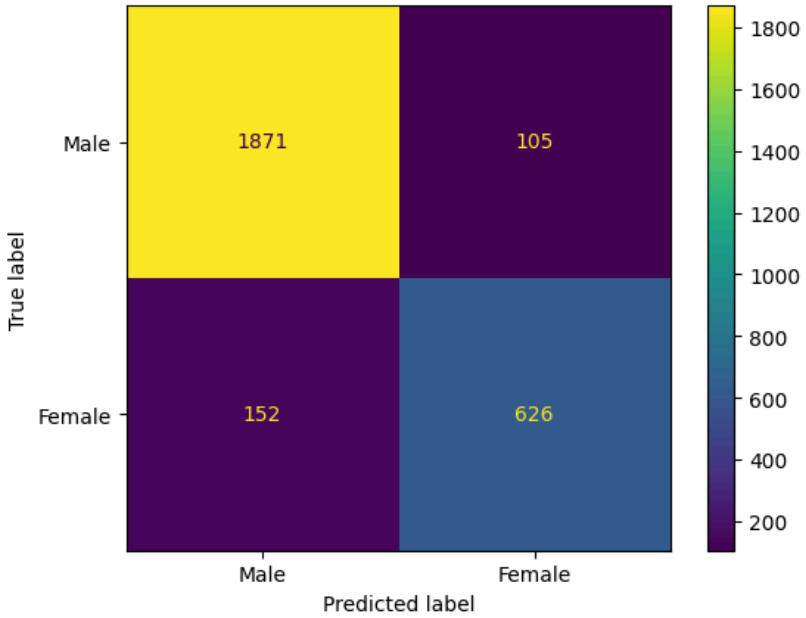
In [19]:
```
print("Classification Report:")
print(classification_rep)
```

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

        male       0.92      0.95      0.94      1976
      female       0.86      0.80      0.83       778

    accuracy                           0.91      2754
   macro avg       0.89      0.88      0.88      2754
weighted avg       0.91      0.91      0.91      2754
```

In [22]:
```
cm_display = ConfusionMatrixDisplay(confusion_matrix = conf_matrix, display_labels = ['Male', 'Female'])
cm_display.plot()
plt.show()
```



In [23]:
```
#torch.save(model.state_dict(), 'custom_model.pth')
```