Convolutional Neural Network - Transfer Learning

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 [1]:

```
import os
import torch
from PIL import Image
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
from torch.optim.lr_scheduler import StepLR
from torchvision import models, transforms
from torch.utils.data import DataLoader, Dataset
from sklearn.metrics import confusion_matrix, classification_report, Confusi
```

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

In [7]:

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

In [8]:

```
# Load pre-trained VGG model
model = models.vgg16(pretrained=True)

for param in model.parameters():
    param.requires_grad = False

# Modify the fully connected layer
num_features = model.classifier[6].in_features

# Custom Classifier
num_features = model.classifier[6].in_features
model.classifier[6] = nn.Sequential(
    nn.Linear(num_features, 256),
    nn.ReLU(),
    nn.Dropout(0.1), # Dropout is applied with probability 0.1 to prevent ov
    nn.Linear(256, 2)  # Output is 2 dimensional (male and female)
)
```

```
C:\Users\aerol\env\facenv\Lib\site-packages\torchvision\models\_utils.py:20
8: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may
be removed in the future, please use 'weights' instead.
  warnings.warn(
C:\Users\aerol\env\facenv\Lib\site-packages\torchvision\models\_utils.py:22
3: UserWarning: Arguments other than a weight enum or `None` for 'weights'
are deprecated since 0.13 and may be removed in the future. The current beh
avior is equivalent to passing `weights=VGG16_Weights.IMAGENET1K_V1`. You c
an also use `weights=VGG16_Weights.DEFAULT` to get the most up-to-date weig
hts.
  warnings.warn(msg)
```

In [9]:

```
model = model.to(device)

# Loss function
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())

scheduler = StepLR(optimizer, step_size=5, gamma=0.1)
```

In [10]:

```
all_preds = []
all_labels = []
```

In [11]:

```
num_epochs = 10
for epoch in range(num_epochs):

    # Training Phase
    model.train()
    running_loss = 0.0
    running_corrects = 0

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

        optimizer.zero_grad()

        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        loss = criterion(outputs, labels)

        loss.backward()
        optimizer.step()

        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels.data)

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

    epoch_loss = running_loss / dataset_sizes['train']
    epoch_acc = running_corrects.double() / dataset_sizes['train']
    scheduler.step()

    print('Train Loss: {:.4f} Acc: {:.4f}'.format(epoch_loss, epoch_acc))
```

```
Epoch [10/10], Phase: train, Batch: [114/131], Loss: 0.3534
Epoch [10/10], Phase: train, Batch: [115/131], Loss: 0.5857
Epoch [10/10], Phase: train, Batch: [116/131], Loss: 0.3874
Epoch [10/10], Phase: train, Batch: [117/131], Loss: 0.3050
Epoch [10/10], Phase: train, Batch: [118/131], Loss: 0.2198
Epoch [10/10], Phase: train, Batch: [119/131], Loss: 0.3936
Epoch [10/10], Phase: train, Batch: [120/131], Loss: 0.2931
Epoch [10/10], Phase: train, Batch: [121/131], Loss: 0.3185
Epoch [10/10], Phase: train, Batch: [122/131], Loss: 0.3314
Epoch [10/10], Phase: train, Batch: [123/131], Loss: 0.3851
Epoch [10/10], Phase: train, Batch: [124/131], Loss: 0.3552
Epoch [10/10], Phase: train, Batch: [125/131], Loss: 0.2988
Epoch [10/10], Phase: train, Batch: [126/131], Loss: 0.3295
Epoch [10/10], Phase: train, Batch: [127/131], Loss: 0.2283
Epoch [10/10], Phase: train, Batch: [128/131], Loss: 0.2966
Epoch [10/10], Phase: train, Batch: [129/131], Loss: 0.2572
Epoch [10/10], Phase: train, Batch: [130/131], Loss: 0.2667
Epoch [10/10], Phase: train, Batch: [131/131], Loss: 0.2706
Train Loss: 0.3451 Acc: 0.8543
```

In [12]:

```
model.eval()
running_loss = 0.0
running_corrects = 0

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

    with torch.no_grad():
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)
        loss = criterion(outputs, labels)

    running_loss += loss.item() * inputs.size(0)
    running_corrects += torch.sum(preds == labels.data)

    all_preds.extend(preds.cpu().numpy())
    all_labels.extend(labels.cpu().numpy())

    batch_loss = loss.item()
    print(f'Validation Batch: [{batch_idx+1}/{len(dataloaders["val"])}], Los
Validation Batch: [1/44], Loss: 0.4852
Validation Batch: [2/44], Loss: 0.3934
Validation Batch: [3/44], Loss: 0.2545
Validation Batch: [4/44], Loss: 0.2638
Validation Batch: [5/44], Loss: 0.2341
Validation Batch: [6/44], Loss: 0.2497
Validation Batch: [7/44], Loss: 0.2504
Validation Batch: [8/44], Loss: 0.2941
Validation Batch: [9/44], Loss: 0.2970
Validation Batch: [10/44], Loss: 0.3511
Validation Batch: [11/44], Loss: 0.1656
Validation Batch: [12/44], Loss: 0.2178
Validation Batch: [13/44], Loss: 0.3902
Validation Batch: [14/44], Loss: 0.2930
Validation Batch: [15/44], Loss: 0.4157
Validation Batch: [16/44], Loss: 0.2052
Validation Batch: [17/44], Loss: 0.1751
Validation Batch: [18/44], Loss: 0.2930
Validation Batch: [19/44], Loss: 0.2530
Validation Batch: [20/44], Loss: 0.2962
Validation Batch: [21/44], Loss: 0.2320
Validation Batch: [22/44], Loss: 0.2634
Validation Batch: [23/44], Loss: 0.2314
Validation Batch: [24/44], Loss: 0.2393
Validation Batch: [25/44], Loss: 0.2293
Validation Batch: [26/44], Loss: 0.2799
Validation Batch: [27/44], Loss: 0.2919
Validation Batch: [28/44], Loss: 0.1738
Validation Batch: [29/44], Loss: 0.3304
Validation Batch: [30/44], Loss: 0.3218
Validation Batch: [31/44], Loss: 0.1321
Validation Batch: [32/44], Loss: 0.3064
Validation Batch: [33/44], Loss: 0.1359
Validation Batch: [34/44], Loss: 0.2352
Validation Batch: [35/44], Loss: 0.1976
Validation Batch: [36/44], Loss: 0.2139
Validation Batch: [37/44], Loss: 0.3062
Validation Batch: [38/44], Loss: 0.1546
Validation Batch: [39/44], Loss: 0.4315
Validation Batch: [40/44], Loss: 0.1391
Validation Batch: [41/44], Loss: 0.2978
Validation Batch: [42/44], Loss: 1.1195
Validation Batch: [43/44], Loss: 1.9106
Validation Batch: [44/44], Loss: 0.7557
```

In [13]:
```
class_names = ['male', 'female']
```

In [14]:
```
conf_matrix = confusion_matrix(all_labels, all_preds)
classification_rep = classification_report(all_labels, all_preds, target_nam

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

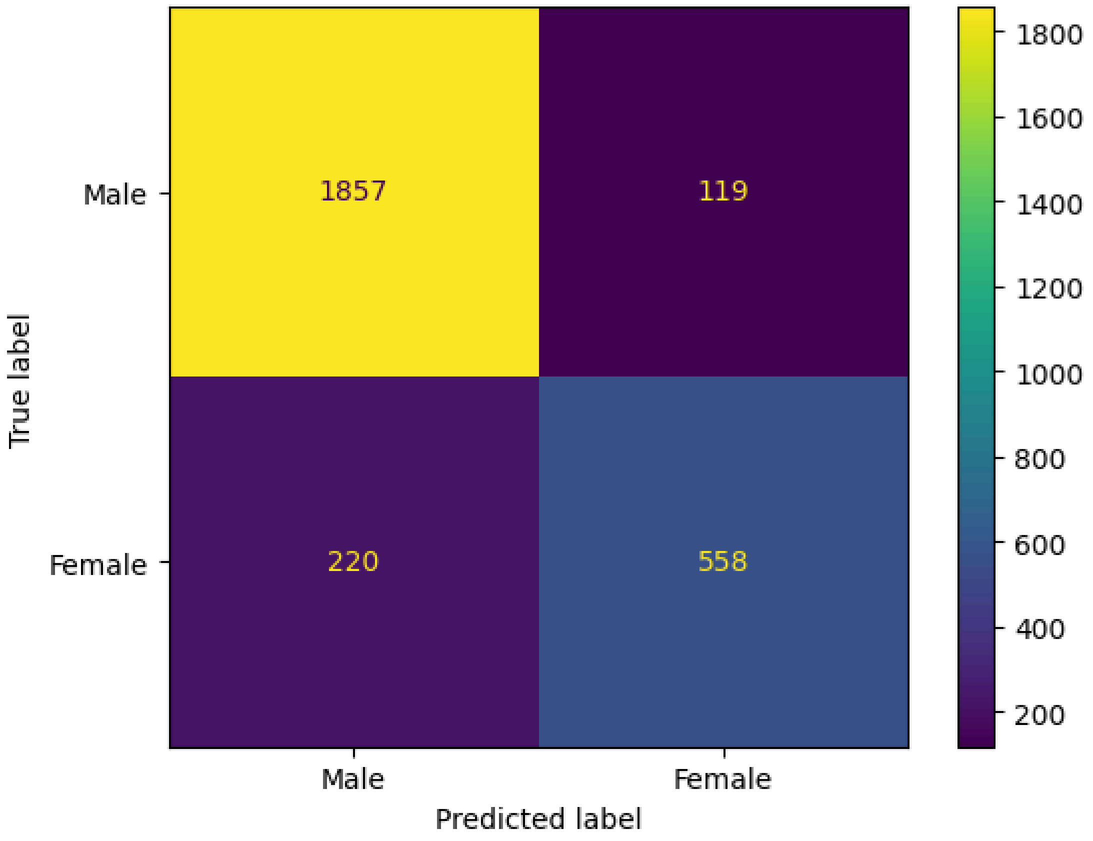
```
Confusion Matrix:
[[1857  119]
 [ 220  558]]
```

In [15]:
```
print("Classification Report:")
print(classification_rep)
```

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

        male       0.89      0.94      0.92      1976
      female       0.82      0.72      0.77       778

    accuracy                           0.88      2754
   macro avg       0.86      0.83      0.84      2754
weighted avg       0.87      0.88      0.87      2754
```

In [16]:
```
cm_display = ConfusionMatrixDisplay(confusion_matrix = conf_matrix, display_
cm_display.plot()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[16], line 3
      1 cm_display = ConfusionMatrixDisplay(confusion_matrix = conf_matrix,
display_labels = ['Male', 'Female'])
      2 cm_display.plot()
----> 3 plt.show()

NameError: name 'plt' is not defined
```



In [17]:
```
torch.save(model.state_dict(), 'vgg_model.pth')
```