# Convolutional Neural Network - Transfer Learning

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

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

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['trai
    'val': datasets.ImageFolder(root=val_dir, transform=transform['val'])
}

dataloaders = {
    'train': DataLoader(image_datasets['train'], batch_size=batch_size, shu
    '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]:
```
# Load Model
model = models.vgg16(pretrained=True)

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

# 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)
)

model = model.to(device)

# Loss function
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.013) # Adam Optimizer

scheduler = StepLR(optimizer, step_size=2, gamma=0.1)
```

```
C:\Users\aerol\env\facenv\Lib\site-packages\torchvision\models\_utils.py:2
08: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and m
ay be removed in the future, please use 'weights' instead.
  warnings.warn(
C:\Users\aerol\env\facenv\Lib\site-packages\torchvision\models\_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for 'weight
s' are deprecated since 0.13 and may be removed in the future. The current
behavior is equivalent to passing `weights=VGG16_Weights.IMAGENET1K_V1`. Y
ou can also use `weights=VGG16_Weights.DEFAULT` to get the most up-to-date
weights.
  warnings.warn(msg)
```

# Training

In [8]:
```
all_preds = []
all_labels = []
```

In [9]:

```
num_epochs = 3
for epoch in range(num_epochs):

    for phase in ['train', 'val']:
        if phase == 'train':
            model.train()
        else:
            model.eval()

        running_loss = 0.0
        running_corrects = 0
        total_batches = len(dataloaders[phase])

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

            optimizer.zero_grad()

            with torch.set_grad_enabled(phase == 'train'):
                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
                loss = criterion(outputs, labels)

                if phase == 'train':
                    loss.backward()
                    optimizer.step()

            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'Epoch [{epoch+1}/{num_epochs}], Phase: {phase}, Bat

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

        print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, e
```

```
Epoch [1/3], Phase: train, Batch: [1/730], Loss: 0.7068
Epoch [1/3], Phase: train, Batch: [2/730], Loss: 4.2294
Epoch [1/3], Phase: train, Batch: [3/730], Loss: 9.2497
Epoch [1/3], Phase: train, Batch: [4/730], Loss: 4.7356
Epoch [1/3], Phase: train, Batch: [5/730], Loss: 0.6188
Epoch [1/3], Phase: train, Batch: [6/730], Loss: 2.7425
Epoch [1/3], Phase: train, Batch: [7/730], Loss: 2.9531
Epoch [1/3], Phase: train, Batch: [8/730], Loss: 1.0421
Epoch [1/3], Phase: train, Batch: [9/730], Loss: 0.5554
Epoch [1/3], Phase: train, Batch: [10/730], Loss: 0.5963
Epoch [1/3], Phase: train, Batch: [11/730], Loss: 0.4012
Epoch [1/3], Phase: train, Batch: [12/730], Loss: 0.6129
Epoch [1/3], Phase: train, Batch: [13/730], Loss: 0.4655
Epoch [1/3], Phase: train, Batch: [14/730], Loss: 0.4602
Epoch [1/3], Phase: train, Batch: [15/730], Loss: 0.3687
Epoch [1/3], Phase: train, Batch: [16/730], Loss: 0.3149
Epoch [1/3], Phase: train, Batch: [17/730], Loss: 0.2650
Epoch [1/3], Phase: train, Batch: [18/730], Loss: 0.3385
Epoch [1/3], Phase: train, Batch: [19/730], Loss: 0.4764
```

In [10]:

```
conf_matrix = confusion_matrix(all_labels, all_preds)
classification_rep = classification_report(all_labels, all_preds, target_nar

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

```
Confusion Matrix:
[[75764 11239]
 [ 9062 78880]]
```

In [11]:

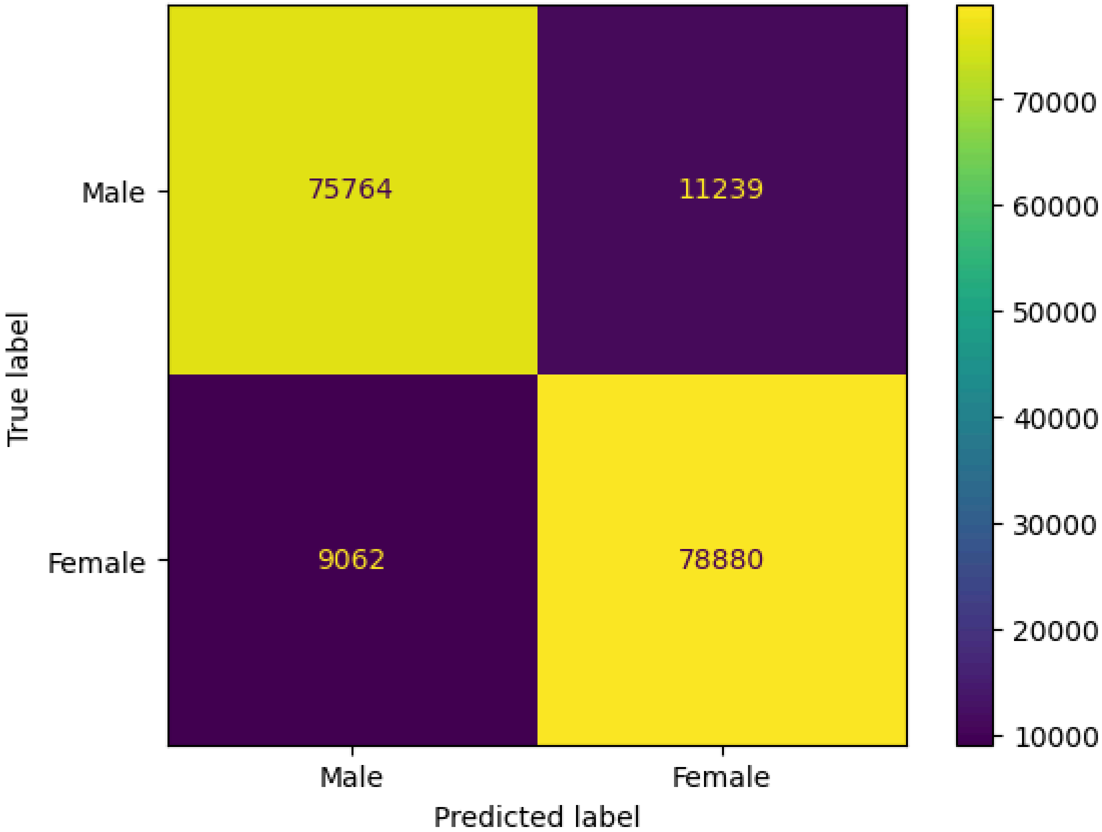
```
print("Classification Report:")
print(classification_rep)
```

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

      female       0.89      0.87      0.88     87003
        male       0.88      0.90      0.89     87942

    accuracy                           0.88    174945
   macro avg       0.88      0.88      0.88    174945
weighted avg       0.88      0.88      0.88    174945
```

In [12]:
```
cm_display = ConfusionMatrixDisplay(confusion_matrix = conf_matrix, display_
cm_display.plot()
plt.show()
```




In [13]:
```
torch.save(model.state_dict(), 'vgg_model.pth')
```

# Inference

In [18]:
```
loaded_model = models.vgg16(pretrained=False)  # Load a new instance of VGG1
num_features = loaded_model.classifier[6].in_features
loaded_model.classifier[6] = nn.Sequential(
    nn.Linear(num_features, 256),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(256, 2)
)
loaded_model.load_state_dict(torch.load('vgg_model.pth'))
loaded_model = loaded_model.to(device)
loaded_model.eval()
```

Out[18]:
```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
de=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mo
de=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Sequential(
      (0): Linear(in_features=4096, out_features=256, bias=True)
      (1): ReLU()
```

```
        (2): Dropout(p=0.3, inplace=False)
        (3): Linear(in_features=256, out_features=2, bias=True)
      )
    )
  )
```

In [20]:

```
def preprocess_image(image_path):
    transform = transforms.Compose([
        transforms.Resize(input_size),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
    image = Image.open(image_path)
    image = transform(image)
    image = image.unsqueeze(0)  # Add batch dimension
    return image.to(device)  # Move the image to GPU if available

image_paths = ['harun.jpg', 'arda.jpg']  # Provide paths to the images
for image_path in image_paths:
    input_image = preprocess_image(image_path)
    with torch.no_grad():
        output = loaded_model(input_image)
        probabilities = torch.softmax(output, dim=1)
        predicted_class = torch.argmax(probabilities, dim=1).item()
        predicted_label = class_names[predicted_class]
        print(f"Image: {image_path}, Predicted Gender: {predicted_label}")
```

```
Image: harun.jpg, Predicted Gender: male
Image: arda.jpg, Predicted Gender: male
```