Import Library

```
import os
from PIL import Image
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torch.optim as optim

from torchvision import transforms
from torchvision.models import vgg16  # or resnet50

import matplotlib.pyplot as plt
import time
import os
import seaborn as sns
plt.ion()
```

```
<contextlib.ExitStack at 0x7930009f35d0>
```

GENDER RECOGNITION

Data Preparation

```
data_path = '/content/drive/MyDrive/FaceRecognition/Dataset'
```

```
from google.colab import drive
drive.mount('/content/drive')
os.listdir(data_path)
```

```
Mounted at /content/drive
['gender_classification.csv',
 'gender_classification.xlsx',
 'class_identity.txt',
 'list_attribute.txt',
 'model_saved',
 'Images']
```

```
if os.path.exists(data_path):
    print(f"Directory found: {data_path}")
    os.listdir(data_path)
else:
    print(f"Directory not found: {data_path}")
```

```
Directory found: /content/drive/MyDrive/FaceRecognition/Dataset
```

```
!find "/content/drive/MyDrive/FaceRecognition" -type d -name Images
```

```
/content/drive/MyDrive/FaceRecognition/Dataset/Images
```

```
import os
import pandas as pd
from PIL import Image
import torch
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms

# Path ke dataset utama
data_path = '/content/drive/MyDrive/FaceRecognition/Dataset'
image_folder = os.path.join(data_path, 'Images')
```

```
image_folder = os.path.join(data_path, 'images')

# Load CSV dan TXT
gender_csv = os.path.join(data_path, 'gender_classification.csv')
identity_txt = os.path.join(data_path, 'class_identity.txt')

identity_df = pd.read_csv(identity_txt, sep=' ', header=None, names=['image_filename', 'identity'])
gender_df = pd.read_csv(gender_csv)
gender_df['gender'] = gender_df['Male'].map({1: 1, 0: 0})

# Validasi file gambar yang memang ada
valid_files = set(os.listdir(image_folder))
identity_df = identity_df[identity_df['image_filename'].isin(valid_files)].reset_index(drop=True)

# format filename dibuat konsisten soalnya file class_identity.txt sama image_filename
# dlm folder gak sama, kemungkinan tanpa padding
min_len = min(len(identity_df), len(gender_df))
identity_df['image_filename'] = identity_df['image_filename'].apply(lambda x: f"{int(x.split('.')[0]):06}.jpg")
gender_df = gender_df.iloc[:min_len].reset_index(drop=True)

combined_df = identity_df.copy()
combined_df['gender'] = gender_df['gender']

## --- Apply robust filtering *before* the split ---
#drop baris yg gendernya NaN
combined_df = combined_df.dropna(subset=['gender'])

##combined_df = pd.merge(identity_df, gender_df[['image_filename', 'gender']], on='image_filename', how='inner')
## kalo pake coding ini tanpa bawahnya akan error karena PyTorch CrossEntropyLoss() butuh angka, bukan string.

#supaya gendernya dalam bentuk angka (0 atau 1)
combined_df = combined_df[combined_df['gender'].isin([0, 1])].reset_index(drop=True)

# Convert ke int
combined_df['gender'] = combined_df['gender'].astype(int)
## --- End of robust filtering ---

print("Cleaned Combined DF:", combined_df.shape)

# Cek hasil
print(identity_df.head())
print(identity_df.shape)
print(combined_df)
## note: buat liat semua combined_df pake print(combined_df)
## buat filter spesifik file pake combined_df.loc[combined_df['image_filename'].isin([...])]
## buat filter dengan condition (misal gender=male) pake combined_df.loc[combined_df['gender'] == 'Male']
```

```
Cleaned Combined DF: (5000, 3)
  image_filename  identity
0     000051.jpg      1446
1     000052.jpg      3896
2     000065.jpg      3046
3     000166.jpg      4328
4     000198.jpg       693
(5000, 2)
     image_filename  identity  gender
0        000051.jpg      1446       1
1        000052.jpg      3896       1
2        000065.jpg      3046       1
3        000166.jpg      4328       0
4        000198.jpg       693       0
...             ...       ...     ...
4995     202320.jpg      9696       0
4996     202340.jpg      8450       1
4997     202347.jpg      7306       1
4998     202357.jpg      9149       0
4999     202566.jpg      7736       0

[5000 rows x 3 columns]
```

```
gender_df.loc[408]  # atau gender_df.iloc[408], datanya csv emang salah, harusnya male bukan female
```

	408
Male	0
gender	0

```
# buat cek data adalah label biner (1 untuk Male, 0 untuk Female), dan emang cuma ada 1 kolom 'Male'.
# trus ditambahin
# gender_df['image_filename'] = gender_df.index.map(lambda x: f"{x+1:06}.jpg")
# gender_df['gender'] = gender_df['Male'].map({1: 'Male', 0: 'Female'})
# supaya bisa merge dengan identity_df. --> buat sambungin label ke filename yang benar
# buat balikin ke asal bisa pake
# original_gender_df = pd.read_csv(gender_csv)
# print(original_gender_df.head())

print(gender_df.columns)
print(gender_df.head())
```

```
Index(['Male', 'gender'], dtype='object')
   Male  gender
0     1       1
1     1       1
2     1       1
3     0       0
4     0       0
```

```
# split the data into train and test sets with a 80:20 ratio
from sklearn.model_selection import train_test_split
train_df, val_df = train_test_split(combined_df, test_size=0.2, random_state=42)

print(f"Train DF size: {len(train_df)}")
print(f"Validation DF size: {len(val_df)}")
```

```
Train DF size: 4000
Validation DF size: 1000
```

Preprocessing

```
import os
import pandas as pd
from PIL import Image
import torch
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms

data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ]),
}

class GenderDataset(Dataset):
    def __init__(self, dataframe, image_folder_path, transform=None):
        # Mengubah self.data menjadi self.dataframe sesuai dengan parameter inisialisasi
        self.dataframe = dataframe
        self.image_folder_path = image_folder_path
        self.transform = transform
```

```
        def __len__(self):
            # Mengembalikan panjang dari self.dataframe
            return len(self.dataframe)
    ## __len__: Mengembalikan jumlah data
    ## Ini dibutuhkan oleh PyTorch DataLoader untuk mengetahui berapa banyak sample di dataset

        def __getitem__(self, idx):
            # Mengambil data dari self.dataframe
            image_filename = self.dataframe.iloc[idx]['image_filename']  # ambil nama file gambar berdasarkan indeks baris k
            gender = self.dataframe.iloc[idx]['gender'] #gender udah guaranteed 0 atau 1
            image_path = os.path.join(self.image_folder_path, image_filename) # Gabungkan path folder dan nama file jadi ful

            try:
              image = Image.open(image_path)
              if image.mode != 'RGB': #image convertion technique to RGB here
                image = image.convert('RGB')

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

              gender_tensor = torch.tensor(int(gender), dtype=torch.long) #Ensure gender is a valid integer (0 or 1) and con

            except FileNotFoundError:
              print(f"Warning: Image file not found: {image_path}. Skipping this item.")
              # If returning None, the DataLoader might raise an error.
              # A common pattern is to return a dummy sample or filter these out beforehand.
              # For simplicity in fixing the NameError, we keep the return None, None but be aware
              # this might cause issues later in training if many files are missing.
              return None, None
            except Exception as e:
              print(f"Warning: Error processing image {image_path}: {e}. Skipping this item.")
               # Similar note as above regarding returning None, None
              return None, None

            return image, gender_tensor

    # Use the defined GenderDataset class instead of the undefined FaceDataset
    # data_transforms is now defined in a previous cell
    train_dataset = GenderDataset(train_df, image_folder, transform=data_transforms['train'])
    val_dataset = GenderDataset(val_df, image_folder, transform=data_transforms['val'])

    # Filter out None values returned by __getitem__ if there were errors
    # This is important if you kept the `return None, None` in __getitem__
    train_dataset = [data for data in train_dataset if data[0] is not None]
    val_dataset = [data for data in val_dataset if data[0] is not None]


    train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)

    dataloaders = {'train': train_loader, 'val': val_loader}
    dataset_sizes = {'train': len(train_dataset), 'val': len(val_dataset)}



    # buat ganti semua gambar jadi format yg diproses model pytorch
    transform = transforms.Compose([
        transforms.Resize((224, 224)),   #--> model pretrained vgg biasanya input 224x224x3
        transforms.ToTensor(),           # ubah pil.image (RGB) jadi tensor dengan shape [3,224,224] inget BGR!!!
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])
    image_folder_path = os.path.join(data_path, 'Images')

    # Buat objek Dataset
    if not train_df.empty:
        train_dataset = GenderDataset(train_df, image_folder_path=image_folder_path, transform=transform)
    else:
        print("Warning: train_df is empty after splitting and filtering. train_dataset will be empty.")
        train_dataset = GenderDataset(pd.DataFrame(columns=train_df.columns), image_folder_path=image_folder_path, transform

    if not val_df.empty:
        val_dataset = GenderDataset(val_df, image_folder_path=image_folder_path, transform=transform)
```

```
else:
     print("Warning: val_df is empty after splitting and filtering. val_dataset will be empty.")
     val_dataset = GenderDataset(pd.DataFrame(columns=val_df.columns), image_folder_path=image_folder_path, transform=tr


## note: pakai os.path.join(data_path, "images") bukan data_path+ '/Images' karena lbh robust & lintas platform

# Buat DataLoader
if len(train_dataset) > 0:
    train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, num_workers=2)
else:
    print("Warning: train_dataset is empty. train_loader will not be created.")
    train_loader = None # Or handle this case appropriately

if len(val_dataset) > 0:
    val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, num_workers=2)
else:
    print("Warning: val_dataset is empty. val_loader will not be created.")
    val_loader = None # Or handle this case appropriately

## note: numworkers buat mengaktifkan multiprocessing saat loading data, buat loading data lebih cepet, terutama kalo tr
## Boleh disesuaikan: num_workers=0 (debug mode), atau num_workers=os.cpu_count() untuk maksimal

# Loader dictionary
dataloaders = {
    'train': train_loader,
    'val': val_loader
}
# Dataset size dictionary
dataset_sizes = {
    'train': len(train_dataset),
    'val': len(val_dataset)
}



transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(), #Mengubah gambar dari PIL.Image ke format tensor PyTorch.
    transforms.Normalize(mean=[0.485, 0.456, 0.406],  # pakai mean & std dari ImageNet
                         std=[0.229, 0.224, 0.225])
])
```

Architecture GOOGLENET

```
# DEFINE MODEL OPTMIZER DAN CRITERION (LOSS FUNCTION)
import torchvision.models as models
import torch.nn as nn
import torch.optim as optim

model = models.googlenet()
model.fc = nn.Linear(1024, 2)  # 1024 adalah output default GoogLeNet, hanya 2 label: male dan female
optimizer = optim.Adam(model.parameters(),lr=0.0001)
criterion = nn.CrossEntropyLoss()

## buat kirim ke GPU
##device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
##model = model.to(device)
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/googlenet.py:47: FutureWarning: The default weight initia
  warnings.warn(
```

Modeling

```
#BUAT FUNGSI TRAIN_MODEL(...)

import time
from torch.autograd import Variable
```

```
import torch
import torchvision.models as models

def train_model(model, dataloaders, dataset_sizes, criterion, optimizer, use_gpu=True, num_epochs=10):
    device = torch.device("cuda" if use_gpu else "cpu")
    model.to(device)

    train_acc_list = []
    val_acc_list = []

    #buat full loop training
    for epoch in range(num_epochs):
        print(f"\nEpoch {epoch+1}/{num_epochs}")
        print("-" * 20)

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()  #Loop kedua: bedain fase training dan validasi.

            running_loss = 0.0 #Variabel untuk menyimpan total loss dan jumlah prediksi benar per epoch.
            correct = 0

            for inputs, labels in dataloaders[phase]:
                inputs, labels = inputs.to(device), labels.to(device)
                optimizer.zero_grad()  #buat reset gradien dari batch sebelumnya

                with torch.set_grad_enabled(phase == 'train'): #nyalain gradien pas training aja, bukan val
                    outputs = model(inputs)   #forward pass --> hasil dari prediksi model

                    # Access the main output tensor from the GoogLeNetOutputs object
                    # GoogLeNet returns a named tuple, access the main logits here.
                    # Depending on the torchvision version, it could be outputs.logits or outputs[0]
                    # Accessing as outputs[0] is generally safer for compatibility.
                    # FIX: Changed models.googlenet.GoogLeNetOutputs to models.GoogLeNetOutputs
                    if isinstance(outputs, models.GoogLeNetOutputs):
                        logits = outputs.logits
                    else:
                        logits = outputs

                    loss = criterion(logits, labels) #hitung selisih prediksi vs label
                    _, preds = torch.max(logits, 1) #ambil prediksi skor tertinggi untuk klasiifikasi

                    if phase == 'train':
                        loss.backward() #buat hitung gradien
                        optimizer.step() #update bobot model sesuai gradien

                running_loss += loss.item() * inputs.size(0)  #Tambah loss batch ke running_loss.
                correct += torch.sum(preds == labels.data) #hitung brp prediksi benar dan tmbh ke correct

            #hitung loss rata2 dan akurasi buat semua data di fase train/val
            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = correct.double() / dataset_sizes[phase]

            #tampilkan hasil loss & akurasi di console.
            print(f"{phase.capitalize()} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}")

            if phase == 'train':
                train_acc_list.append(round(epoch_acc.item() * 100, 1))
            else:
                val_acc_list.append(round(epoch_acc.item() * 100, 1))

    print("\ntrain_acc =", train_acc_list)
    print("test_acc  =", val_acc_list)

    return model



# Buang semua baris dengan NaN
combined_df = combined_df.dropna(subset=['gender'])
```

```
# Pastikan hanya angka 0 atau 1
combined_df = combined_df[combined_df['gender'].isin([0, 1])]
combined_df['gender'] = combined_df['gender'].astype(int)

# Reset index
combined_df = combined_df.reset_index(drop=True)

print("Cleaned Combined DF:", combined_df.shape)
```

```
Cleaned Combined DF: (5000, 3)
```

```
#DEFINE FUNGSI TRAIN
use_gpu = torch.cuda.is_available()
print(f"Using GPU: {use_gpu}")

model = train_model(model, dataloaders, dataset_sizes, criterion, optimizer, use_gpu, 10)
```

```
Using GPU: True

Epoch 1/10
----------
Train Loss: 0.7025 Acc: 0.5623
Val Loss: 0.6859 Acc: 0.5720

Epoch 2/10
----------
Train Loss: 0.6862 Acc: 0.5785
Val Loss: 0.6892 Acc: 0.5770

Epoch 3/10
----------
Train Loss: 0.6832 Acc: 0.5870
Val Loss: 0.6918 Acc: 0.5660

Epoch 4/10
----------
Train Loss: 0.6786 Acc: 0.5920
Val Loss: 0.6877 Acc: 0.5770

Epoch 5/10
----------
Train Loss: 0.6786 Acc: 0.5925
Val Loss: 0.6904 Acc: 0.5690

Epoch 6/10
----------
Train Loss: 0.6791 Acc: 0.5837
Val Loss: 0.6858 Acc: 0.5720

Epoch 7/10
----------
Train Loss: 0.6755 Acc: 0.5885
Val Loss: 0.6880 Acc: 0.5660

Epoch 8/10
----------
Train Loss: 0.6757 Acc: 0.5933
Val Loss: 0.7096 Acc: 0.5670

Epoch 9/10
----------
Train Loss: 0.6739 Acc: 0.5953
Val Loss: 0.6908 Acc: 0.5700

Epoch 10/10
----------
Train Loss: 0.6730 Acc: 0.5995
Val Loss: 0.6889 Acc: 0.5660

train_acc = [56.2, 57.9, 58.7, 59.2, 59.2, 58.4, 58.9, 59.3, 59.5, 60.0]
test_acc  = [57.2, 57.7, 56.6, 57.7, 56.9, 57.2, 56.6, 56.7, 57.0, 56.6]
```

PAKAI DATA AUGMENTATION KARENA OVERFITTING

```
from torchvision import transforms
from torchvision import models

train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.1, contrast=0.1),  # opsional
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

val_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])



model = models.googlenet(pretrained=True)
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/googlenet-1378be20.pth" to /root/.cache/torch/hub/checkpoints/goog
100%|██████████| 49.7M/49.7M [00:00<00:00, 189MB/s]
```

```
import torch.nn as nn  #GoogLeNet butuh layer classifier yang disesuaikan

num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)


train_dataset = GenderDataset(train_df, image_folder, transform=train_transform)
val_dataset = GenderDataset(val_df, image_folder, transform=val_transform)

train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True, num_workers=2)
val_loader = DataLoader(val_dataset, batch_size=8, shuffle=False, num_workers=2)

dataloaders = {'train': train_loader, 'val': val_loader}

dataset_sizes = {
    'train': len(train_dataset),
    'val': len(val_dataset)
}



model = train_model(model, dataloaders, dataset_sizes, criterion, optimizer, use_gpu, 10)
```

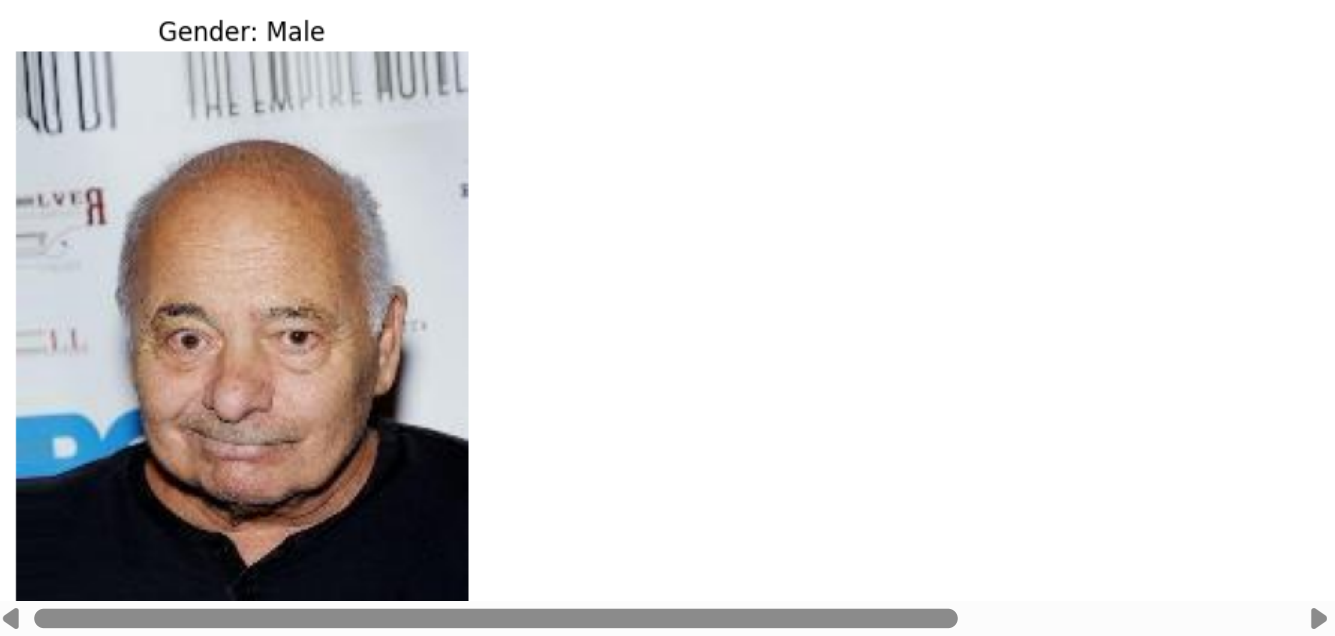
```
Epoch 1/10
--------------------
Train Loss: 0.6877 Acc: 0.5797
Val Loss: 0.6946 Acc: 0.5770

Epoch 2/10
--------------------
Train Loss: 0.6827 Acc: 0.5837
Val Loss: 0.6837 Acc: 0.5750

Epoch 3/10
--------------------
Train Loss: 0.6793 Acc: 0.5930
Val Loss: 0.7207 Acc: 0.5770

Epoch 4/10
--------------------
Train Loss: 0.6795 Acc: 0.5885
Val Loss: 0.6824 Acc: 0.5760
```

```
Epoch 5/10
--------------------
Train Loss: 0.6780 Acc: 0.5913
Val Loss: 0.6890 Acc: 0.5770

Epoch 6/10
--------------------
Train Loss: 0.6785 Acc: 0.5913
Val Loss: 0.6904 Acc: 0.5770

Epoch 7/10
--------------------
Train Loss: 0.6788 Acc: 0.5887
Val Loss: 0.6860 Acc: 0.5570

Epoch 8/10
--------------------
Train Loss: 0.6788 Acc: 0.5903
Val Loss: 0.6812 Acc: 0.5820

Epoch 9/10
--------------------
Train Loss: 0.6799 Acc: 0.5913
Val Loss: 0.7005 Acc: 0.5630

Epoch 10/10
--------------------
Train Loss: 0.6778 Acc: 0.5897
Val Loss: 0.7325 Acc: 0.5770

train_acc = [58.0, 58.4, 59.3, 58.9, 59.1, 59.1, 58.9, 59.0, 59.1, 59.0]
test_acc  = [57.7, 57.5, 57.7, 57.6, 57.7, 57.7, 55.7, 58.2, 56.3, 57.7]
```

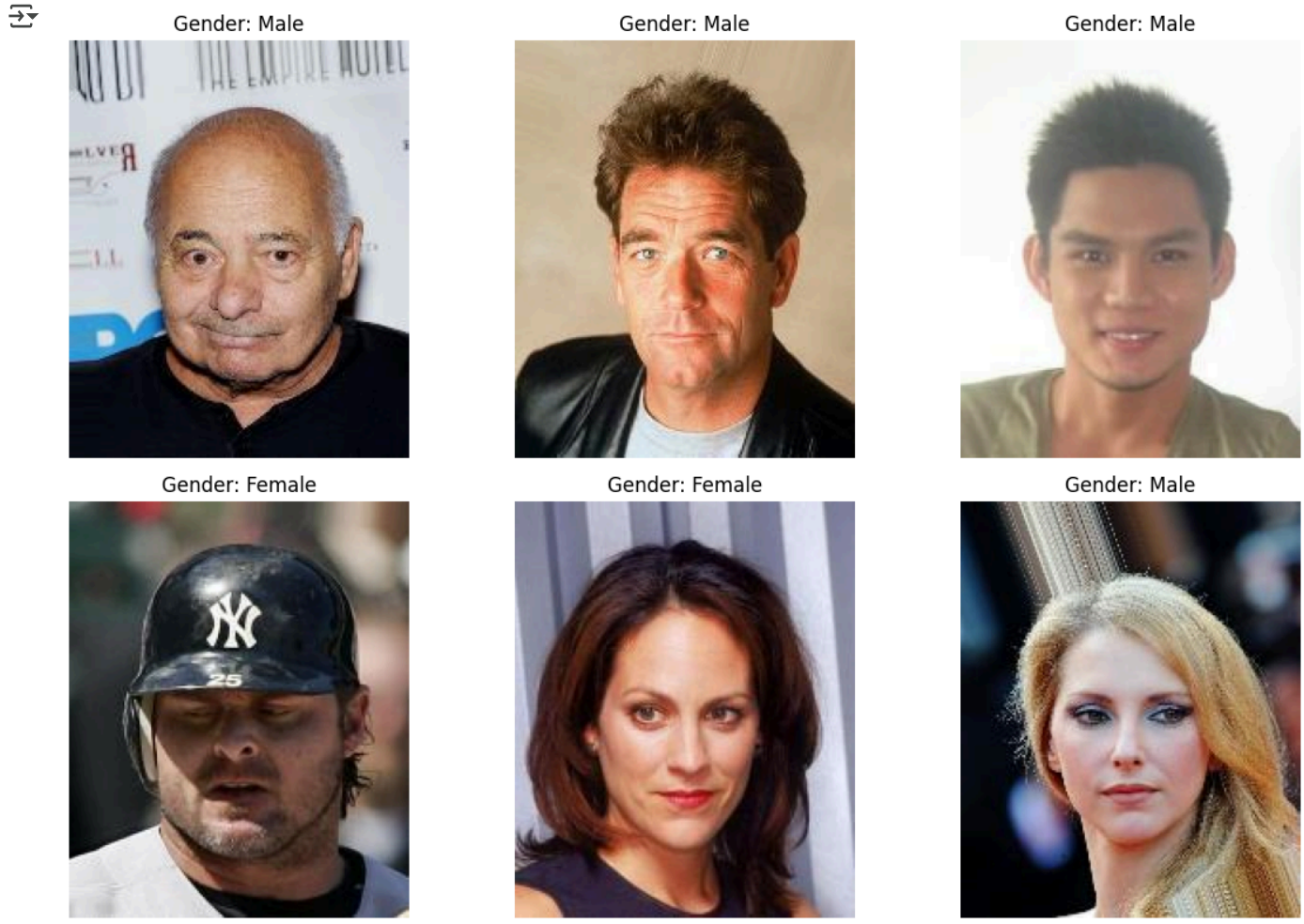
Evaluation

```
from sklearn.metrics import classification_report, confusion_matrix
import torch
import numpy as np

def evaluate_model(model, test_loader, target_labels=['Female', 'Male']):
    model.eval()
    device = next(model.parameters()).device

    all_preds = []
    all_labels = []

    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            _, preds = torch.max(outputs, 1)

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

    all_preds = np.array(all_preds)
    all_labels = np.array(all_labels)

    print("Classification Report:")
    print(classification_report(all_labels, all_preds, target_names=target_labels))

    print("Confusion Matrix:")
    print(confusion_matrix(all_labels, all_preds))

    return all_labels, all_preds


from PIL import Image
import matplotlib.pyplot as plt
import os
```

```
# Ambil baris pertama dari identity_df
sample_row = combined_df.iloc[0]

# Path gambar
image_folder = os.path.join(data_path, 'Images')
img_path = os.path.join(image_folder, sample_row['image_filename'])

# Buka gambar
image = Image.open(img_path)

# Konversi label numeric ke teks
gender_label = 'Male' if sample_row['gender'] == 1 else 'Female'

# Tampilkan gambar dan label
plt.imshow(image)
plt.axis('off')
plt.title(f"Gender: {gender_label}")
plt.show()
```

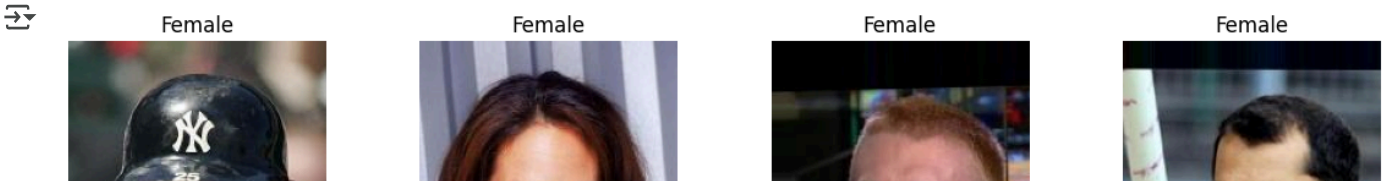



```
fig, axes = plt.subplots(2, 3, figsize=(12, 8))

for i, ax in enumerate(axes.flat):
    if i < len(combined_df):
      row = combined_df.iloc[i]
      img_path = os.path.join(image_folder, row['image_filename'])
      image = Image.open(img_path)

      gender_label = 'Male' if row['gender'] == 1 else 'Female'
      ax.imshow(image)
      ax.set_title(f"Gender: {gender_label}")
      ax.axis('off')
    else:
      break

plt.tight_layout()
plt.show()
```




```
female_df = combined_df[combined_df['gender'] == 0].reset_index(drop=True)
fig, axes = plt.subplots(2, 3, figsize=(12, 8))

for i, ax in enumerate(axes.flat):
  if i < len(female_df):
    row = female_df.iloc[i]
    img_path = os.path.join(image_folder, row['image_filename'])
    image = Image.open(img_path)
    ax.imshow(image)
    ax.set_title("Female")
    ax.axis('off')
  else:
    break

plt.tight_layout()
plt.show()
```


```
female_df = combined_df[combined_df['gender'] == 0].reset_index(drop=True)

fig, axes = plt.subplots(3, 4, figsize=(12, 9))  # tampilkan 12 sample
for i, ax in enumerate(axes.flat):
    row = female_df.iloc[i]
    img_path = os.path.join(image_folder, row['image_filename'])
    image = Image.open(img_path)
    ax.imshow(image)
    ax.set_title("Female")
    ax.axis('off')

plt.tight_layout()
plt.show()
```


Female
Female
Female
Female